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+

Wszechnica Poranna:

Bazy danych

Mechanizmy wewnętrzne baz danych

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.

Mechanizmy wewnętrzne baz danych

Rodzaj zajęć: Wszechnica Poranna
Tytuł: Mechanizmy wewnętrzne baz danych
Autor: mgr inż. Andrzej Ptasznik

Redaktor merytoryczny: prof. dr hab. Maciej M Sysło

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 2010

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.

Mechanizmy wewnętrzne baz danych

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

Streszczenie

Zakres wykładu obejmuje wybrane zagadnienia dotyczące mechanizmów dostępnych w ramach bazy danych. Omówione zostaną więzy integralności referencyjnej wraz z zaprezentowaniem przykładów i dobrych praktyk ich dotyczących. Drugim zagadnieniem będą transakcje – zapoznamy się z istotą transakcyjności, trybami tworzenia transakcji oraz z poziomami izolacji i ich specyfiką. W ramach omawiania wyzwalaczy zademonstrowane zostaną ich rodzaje oraz opisana zostanie ich specyfika. Zaprezentowane zostaną także procedury składowane oraz funkcje użytkownika. Kolejnym zagadnieniem będą indeksy. Zapoznamy się z fizyczną organizacją danych w SQL Serverze 2008 i z jej konsekwencjami. Omówiona zostanie też struktura różnych rodzajów indeksów, a także zaprezentowany zostanie proces optymalizacji zapytania z wykorzystaniem indeksu pokrywającego.

W ramach warsztatów wykonane zostaną ćwiczenia pokazujące sposób konfigurowania i działanie wybranych mechanizmów. Część ćwiczeń będzie wykonywana wspólnie z prowadzącym warsztaty a część będzie pokazem wykonywanym przez prowadzącego.

Spis treści

Wykład

Warsztaty

1 WPROWADZENIE

Na hasło „relacyjna baza danych" pierwszym skojarzeniem jest zwykle tabela lub kilka tabel. Niestety w tym miejscu kończy się wiedza części programistów i w tworzonych przez nich aplikacjach baza danych jest wykorzystywana jedynie jako prymitywna składnica danych. Wszelkie operacje weryfikacji poprawności danych, zapewnienia im spójności i sensowności oraz samo przetwarzanie danych odbywa się w ramach logiki aplikacji. Taka architektura aplikacji nie jest dobrym pomysłem i bardzo szybko okazuje się być niewystarczająca a wręcz szkodliwa lub uniemożliwiająca korzystanie z aplikacji ze względu na skrajnie kiepską wydajność oraz, co ważniejsze, na błędy i niespójności w danych przechowywane w bazie.

Co więc można zrobić, żeby zadbać nieco bardziej o cenne dane, które chcemy gromadzić w bazie? Przede wszystkim należy nieco bliżej poznać możliwości, które są oferowane przez relacyjne bazy danych w zakresie zapewnienia spójności przechowywanym danym. W ramach niniejszego wykładu postaramy się przybliżyć kilka takich mechanizmów, a także zaprezentować sporą ilość dodatkowych informacji, które pozwolą spojrzeć na bazę danych, jako na twór, który można w bardzo szerokim zakresie kształtować, wzbogacać jego funkcjonalność, definiować najprzeróżniejsze ograniczenia mające zastosowanie do przechowywanych danych tak, aby móc spełnić wymagania stawiane tworzonemu systemowi informatycznemu oraz zapewnić bezpieczeństwo i spójność przechowywanym danym.

Zakres wykładu obejmuje wybrane zagadnienia dotyczące mechanizmów dostępnych w ramach bazy danych. Omówione zostaną więzy integralności referencyjnej wraz z zaprezentowaniem przykładów i dobrych praktyk ich dotyczących. Drugim zagadnieniem będą transakcje – zapoznamy się z istotą transakcyjności, trybami tworzenia transakcji oraz z poziomami izolacji i ich specyfiką. W ramach omawiania wyzwalaczy zademonstrowane zostaną poszczególne ich rodzaje oraz opisana zostanie ich specyfika. Zaprezentowane zostaną także procedury składowane oraz funkcje użytkownika. Kolejnym zagadnieniem będą indeksy. Zapoznamy się z fizyczną organizacją danych w SQL Server 2008 i z jej konsekwencjami. Omówiona zostanie też struktura różnych rodzajów indeksów, a także zaprezentowany zostanie proces optymalizacji zapytania z wykorzystaniem indeksu pokrywającego. Istotnym zagadnieniem są też kwestie związane z wykonywaniem kopii bezpieczeństwa baz danych oraz z odtwarzaniem danych z takich kopii.

Rozpoczniemy od mechanizmu zwanego więzami integralności referencyjnej. Jest on istotnym mechanizmem, gdyż umożliwia definiowanie relacji pomiędzy tabelami oraz zapewnia spójność takiej relacji.

2 REGUŁY I OGRANICZENIA

Każda kolumna w tabeli relacyjnej musi mieć określony typ danych. Deklaracja typu jest pierwszym krokiem na drodze zapewnienia poprawności przechowywanych danych. MS SQL Server 2008 udostępnia zbiór predefiniowanych typów danych. Poniżej podajemy przykładowe typy danych w SQL Server 2008.

Dla danych znakowych

- ***char(n)*** – ciąg n znaków o stałej długości (np. jeżeli kolumna ma określony typ char(25) i wpiszemy słowo „kot", to zostanie ono zapisane pomocą 25 znaków – uzupełnione spacjami);
- ***varchar(n)*** – ciąg n znaków o zmiennej długości (np. jeżeli kolumna ma określony typ varchar(25) i wpiszemy słowo „kot", to zostanie ono zapisane za pomocą 3 znaków);
- ***varchar(max)*** – ciąg znaków o zmiennej długości do 2 GB.

W tym miejscu można spróbować odpowiedzieć na następujące pytanie: Skoro typ ***char*** w porównaniu z ***varchar*** wykorzystuje więcej pamięci do zapisywania danych (uzupełnianie spacjami), to jakie korzyści możemy osiągnąć w przypadku wykorzystania typu ***char***.

Dla danych liczbowych – liczby całkowite

- ***tinyint*** – liczba całkowita z zakresu [0, 255], przechowywana w jednym bajcie;
- ***smallint*** – liczba całkowita z zakresu [–32768, 32767], przechowywana na dwóch bajtach;
- ***int*** – liczba całkowita z zakresu [–2147483648, 2147483647], przechowywana na czterech 4 bajtach;
- ***bigint*** – liczba całkowita z zakresu [–9223372036854775808, 922337203685477580], przechowywana na ośmiu bajtach.

Dla danych liczbowych – liczby z ułamkiem

- ***real, float*** – do zapisywania liczb zmiennopozycyjnych;
- ***decimal, numeric*** – do zapisywania liczb zmiennopozycyjnych o określonej precyzji;
- ***money*** – do zapisywania liczb wyrażających kwoty pieniężne.

Dla danych – daty i czasu

- ***date*** – do zapisywania dat, np. 2009-08-22;
- ***time*** – do zapisywania czasu, np. 19:22:07.2345644;
- ***datetime*** – do zapisywania łącznie daty i czasu, np. 2009-08-22 19:22:07.2345644.

Typy różne

- ***bit*** – do zapisywania wartości logicznych (true, false lub 0,1);
- ***varbinary(n)*** – do zapisywania danych binarnych o długości n bajtów;
- ***varbinary(max)*** – do zapisywania danych binarnych o długości do 2 GB (np. obrazy, dźwięki);
- ***timestamp*** – specjalny znacznik który automatycznie zmienia swoja wartość przy modyfikacji wiersza.

Nie wymieniliśmy wszystkich dostępnych typów danych, widać jednak, że dostępny jest dość duży zbiór typów danych i od decyzji projektanta zależy właściwy ich wybór. Krótkie podsumowanie:

- Każda kolumna w tabeli musi mieć określony typ danych, w jakim będą w tej kolumnie zapisywane dane.
- Decyzja o wyborze odpowiedniego typu danych jest pierwszym etapem zapewnienia spójności danych.
- Wybór typu jest równoznaczny z określeniem dziedziny wartości dla danych zapisywanych w danej kolumnie.

Kolejnym mechanizmem związanym z zapewnieniem spójności i integralności danych są definicje kluczy. W każdej tabeli relacyjnej powinien być zdefiniowany klucz podstawowy – taka definicja zapewnia, że każda wartość w kolumnie klucza podstawowego musi przyjąć inną wartość. W systemach zarządzania bazami danych (SZBD) istnieją mechanizmy nadające kolumnom klucza podstawowego automatycznie unikatowe wartości (autonumeracja), a samo zdefiniowanie kolumny jako klucza podstawowego wymusza jednoznaczność wprowadzanych danych, czyli system nie zezwoli na to, żeby w danej tabeli pojawiły się dwie identyczne wartości w kolumnie klucza podstawowego. Można także wymusić jednoznaczność kolumn, które nie są kluczem podstawowym, czyli definiować tzw. **klucze potencjalne.**

Omawiana wcześniej deklaracja typu określa dziedzinę wartości dla kolumny, ale często jest to dziedzina zbyt szeroka, czyli nie wszystkie wartości z tak określonej dziedziny możemy uznać za poprawne z punktu widzenia zawartości analizowanej kolumny. Dodatkowym utrudnieniem mogą być także zależności logiczne pomiędzy danymi zapisanymi w różnych kolumnach jednego wiersza. Jeżeli jesteśmy w stanie zdefiniować takie zależności, to korzystając z mechanizmu, dostarczanego przez SZBD, definiowania reguł poprawności dla kolumny lub wiersza. Problemy związane z regułami poprawności przeanalizujemy na przykładzie kolumny ***pesel*** z pokazanej na rysunku 1 przykładowej tabeli.

Rysunek 1.
Tabela uczniowie

Załóżmy, że dla kolumny ***pesel*** został zdefiniowany typ danych ***char(11)***. Wydaje się, że jest to definicja poprawna, ale rodzi się wiele problemów związanych z zapewnieniem poprawności danych zapisywanych w tej kolumnie, czyli musimy wymusić, żeby każda wartość zapisana w tej kolumnie była poprawnym numerem pesel.

Z punktu widzenia deklaracji typu ***char(11)***, poprawnymi wartościami są wszystkie ciągi znakowe o długości nie przekraczającej 11 znaków i w tym momencie widzimy, jak daleko jeszcze do pełnej poprawności.

Gdybyśmy pozostali na takim zdefiniowaniu tej kolumny, to za poprawne dane mogłyby uchodzić nawet tak bezsensowne dane: 'Ala ma kota', 'W45991AS', 'brak Pesel' - każdy z tych przykładowych ciągów znakowych jest poprawny, ponieważ nie przekracza 11 znaków.

Zadanie 1. Ograniczyć definicję typu tak, żeby jako poprawne były traktowane tylko ciągi składające się z dokładnie jedenastu znaków. W tym celu można skorzystać z mechanizmu definiowania ograniczeń. Ograniczenie dziedziny wartości sprowadza się do zdefiniowania wyrażenia logicznego, które, jeśli jest spełnione, uznaje dane za poprawne. W naszym przypadku takie wyrażenie mogłoby mieć następująca postać:

ile_znakow(pesel) = 11

Załóżmy, że istnieje funkcja o nazwie ***ile_znakow***, której jako parametr przekazujemy ciąg znaków (w naszym przypadku zawartość kolumny ***pesel***), a w wyniku otrzymujemy liczbę znaków, z których składa się przekazany parametr. Jeżeli funkcja o takiej nazwie nie istnieje w SZBD, to możemy skorzystać z istniejącej funkcji o innej nazwie wykonującej podobne działanie albo utworzyć własną funkcję. Ponieważ w ramach tego wykładu zajmujemy się głównie problemami, to szczegóły realizacji są w tym przypadku mniej istotne. Możemy jedynie zapewnić, że praktycznie w każdym SZBD można taki warunek zdefiniować. W tym miejscu uznajemy, że potrafimy zdefiniować taką regułę i ... to dopiero początek długiej drogi, ponieważ po tej definicji za poprawne zapisy uznane zostaną następujące ciągi znakowe:

'aswedfcxsdr' – bo zawiera dokładnie 11 znaków,
'a234543234j' – bo też zawiera dokładnie 11 znaków.

Zadanie 2. Zapewnić, że dane w kolumnie ***pesel*** składają się z dokładnie 11 cyfr. W tej sytuacji zadanie sprowadza się do uściślenia naszego wcześniejszego wyrażenia. Posłużymy się w tej sytuacji również hipotetyczna funkcją. Nasze wyrażenie logiczne mogłoby mieć następującą postać :

ile_znakow(pesel) = 11 and ile_cyfr(pesel)=11

To wyrażenie zapewni, że jako poprawne zostaną uznane tylko ciągi znaków złożone z 11 cyfr i ... byłoby już prawie dobrze, gdyby nie fakt, że w numerze ***pesel*** ostatnia cyfra nie jest cyfrą przypadkową, tylko tak zwaną **sumą kontrolną**. Mechanizmy cyfr kontrolnych są stosowane przez różnego typu identyfikatory (pesel, nip, numer bankowy, numer dowodu osobistego) dla zapobiegania przypadkowym błędom przy wprowadzaniu danych. Jeżeli chcemy traktować bazę poważnie, to powinniśmy także zapewnić sprawdzanie cyfry kontrolnej, a to polega na tym, że według pewnego algorytmu korzystając z pierwszych dziesięciu cyfr numeru Pesel wykonujemy obliczenia, których wynik musi być równy ostatniej cyfrze numeru Pesel i tylko wtedy taki Pesel jest poprawny.

Zadanie 3. Zapewnić sprawdzenie poprawności cyfry kontrolnej, czyli rozbudować wyrażenie sprawdzające poprawność danych i zgodnie z przyjętymi zasadami odnośnie założenia istnienia takich funkcji, które są w danej chwili potrzebne. Zmodyfikowana postać takiego wyrażenia mogłaby wyglądać następująco:

ile_znakow(pesel) = 11 and ile_cyfr(pesel)=11
and ostatnia_cyfra(pesel)=cyfra_kontrolna(pesel)

Nasza definicja staje się coraz bardziej złożona, ale po jej określeniu to SZBD będzie sprawdzał i wymuszał poprawność zapisywanych danych. Napracowaliśmy się dużo i ... nagle olśnienie, że przecież pierwsze 6 cyfr w numerze Pesel także nie może być dowolnych ale musi odpowiadać dacie urodzenia, tym bardziej, ze w naszej tabeli obok kolumny ***pesel*** jest także kolumna ***Data_urodzenia*** i niedopuszczalne byłoby zaniechanie synchronizacji miedzy tymi danymi.

Zadanie 4. Zapewnić, żeby numer Pesel był zgodny z zapisaną w tym samym wierszu datą urodzenia. Zgodnie z naszą tradycją rozbudujemy wyrażenie logiczne o kolejny składnik, który będzie odpowiedzialny za sprawdzenie zgodności numeru pesel z datą urodzenia. Po modyfikacji wyrażenie logiczne przyjmie następującą postać :

ile_znakow(pesel) = 11 and ile_cyfr(pesel)=11
and dziesiata_cyfra(pesel)=cyfra_kontrolna(pesel) and
data_z_pesel(pesel)=data_urodzenia

W tym miejscu chcielibyśmy zwrócić uwagę, że po raz pierwszy wyrażenie odwołuje się do różnych kolumn tabeli (oczywiście ma to sens w obrębie jednego konkretnego wiersza). Moglibyśmy pewnie już spocząć na laurach, gdyby nie jeszcze jedna wiadomość mówiąca o tym, że dziesiąta cyfra numeru Pesel także nie jest przypadkowa, gdyż jest to cyfra określająca płeć (parzysta kobiety, nieparzysta mężczyźni) a pamiętamy, że w naszej tabeli jest kolumna o dziwnej nazwie ***czy_chlopak***, w której zapisujemy dane określające płeć.

Zadanie 5. Zapewnić sprawdzenie poprawności oznaczenia płci w numerze Pesel. W tej sytuacji została nam, mamy nadzieje ostatnia modyfikacji naszej reguły poprawności do następującej postaci :

ile_znakow(pesel) = 11 and ile_cyfr(pesel)=11
and dziesiata_cyfra(pesel)=cyfra_kontrolna(pesel) and
data_z_pesel(pesel)=data_urodzenia and
sprawdz_plec(czy_chlopak, pesel)=1

W tym przypadku założyliśmy, że istnieje funkcja o nazwie ***sprawdz_plec***, która przyjmuje dwa parametry (***czy_chlopak*** i ***pesel***) i jeżeli zwróci wartość 1, to znaczy, że płeć w numerze ***pesel*** jest prawidłowa.

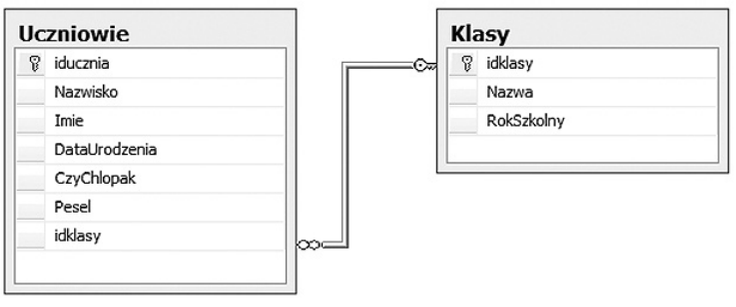
Przyznać trzeba, że rozważany problem został rozwiązany w zbyt skomplikowany sposób. Równie dobrze można było założyć istnienie jednej funkcji, która realizuje wszystkie nasze zadania – funkcja o nazwie ***sprawdz_pesel***, której przekażemy trzy parametry (***pesel***, ***data-urodzenia***, ***czy_chlopak***) i jeżeli taka funkcja zwróci wartość 1, to uznamy dane zapisane w wierszu za poprawne:

sprawdz_pesel(pesel, data_urodzenia, czy_chlopak) = 1

Niezależnie od ostatecznej postaci wyrażenia wymuszającego poprawność, w trakcie omawiania problemów związanych z poprawnością numeru Pesel wykazaliśmy, że zagadnienia zapewnienia spójności i integralności danych są złożone, ale jednocześnie bardzo istotne, gdyż tylko wymuszenie poprawności danych może gwarantować przydatność całej bazy danych.

3 WIĘZY INTEGRALNOŚCI REFERENCYJNEJ

Przy projektowaniu bazy danych standardową praktyką jest doprowadzenie struktury bazy do tzw. **trzeciej postaci normalniej**. Prowadzi to do powstania większej liczby mniejszych (czyli zawierających mniej kolumn) tabel. Żeby zapewnić spójność danych w takiej sytuacji potrzebny jest specjalny mechanizm, który po zdefiniowaniu reguł powiązań miedzy tabelami będzie pilnował ich spójności. Właśnie do tego celu stworzony został jeden z rodzajów ograniczeń – **klucz obcy**. Za jego pomocą możemy w wygodny sposób definiować reguły spójności danych pomiędzy tabelami. Problem ten omówimy odnosząc się do przykładu pokazanego na rysunku 1.



Rysunek 2.
Przykład powiązanych tabel

Tabela Uczniowie zawiera kolumnę idklasy, która pełni rolę klucza obcego wiążącego wiersz z tabeli Uczniowie z odpowiednim wierszem z tabeli Klasy. Takie zależności są podstawą w projektowaniu baz danych opartych na modelu relacyjnym pod warunkiem, że zapewnimy poprawność polegającą na tym, że wartości klucza obcego zawsze znajdą swój odpowiednik w tabeli powiązanej. Mówiąc prosto, wartości zapisywane w kolumnie idklasy tabeli Uczniowie muszą pochodzić ze zbioru wartości idklasy występującego w tabeli Klasy. W pokazanym przykładzie możliwe są operacje wykonywane na danych w tabelach Uczniowie i Klasy, które mogą spowodować problemy z właściwą interpretacją danych:

- Wstawianie nowego wiersza do tabeli Uczniowie – w trakcie wstawiania danych nowego wiersza można próbować zapisać taką wartość w kolumnie idklasy, która nie występuje w tabeli Klasy. Jest to sytuacja błędna, ponieważ tak zapisany uczeń byłby powiązany z klasą, która nie istnieje.
- Usuwanie wiersza z tabeli Klasy – usunięcie wiersza z tabeli Klasy (jeżeli są uczniowie powiązani z tą klasa) spowodowałoby utratę możliwości powiązania grupy uczniów z właściwą klasą.
- Modyfikacja kolumny Idklasy w tabeli Uczniowie – zmiana wartości idklasy na wartość, która nie występuje w tabeli Klasy, spowoduje utratę możliwości powiązania tego ucznia z właściwą klasą.

Przy tworzeniu ograniczeń typu klucz obcy należy wspomnieć o kaskadowych więzach integralności referencyjnej. Ich istota polega na zdefiniowaniu, w jaki sposób mają się zachować wiersze (i kolumny) klucza obcego w reakcji na usunięcie (lub zmodyfikowanie) wiersza (lub wartości klucza podstawowego). Można wybrać jeden z czterech wariantów takiej reakcji dla każdej z dwóch opcji (ON DELETE i ON UPDATE):

- No Action – jest to domyślny wariant, który nie powoduje podjęcia jakichkolwiek działań w związku z usunięciem czy zmodyfikowaniem wiersza, do którego odwołuje się klucz obcy.
- Cascade – powoduje kaskadowe usunięcie wierszy odwołujących się poprzez klucz obcy do usuwanego wiersza, lub modyfikację wartości kolumn klucza obcego w przypadku zmodyfikowania wartości klucza podstawowego, do którego się on odwołuje. Jest to opcja z jednej strony bardzo wygodna, gdyż zwalnia nas z „ręcznego" usuwania wierszy skojarzonych przez klucz obcy. Z drugiej strony zaś, przy odrobinie pecha można wyczyścić niechcący sporą część danych jednym niewinnym poleceniem usunięcia jednego wiersza z tabeli.
- Set Null – polega na tym, że w przypadku usunięcia wiersza do którego odwoływał się klucz obcy, kolumnom tegoż klucza przypisywana jest wartość null. Żeby opcja ta zadziałała, kolumny klucza obcego musza dopuszczać wartość null.
- Set Default – działanie zbliżone do Set null, z tą różnicą, że kolumny klucza obcego są ustawiane na wartość domyślną. Żeby opcja ta zadziałała, kolumny klucza muszą mieć określona wartość domyślną lub dopuszczać wartość null (na którą będą ustawione w przypadku braku wartości domyślnej).

Sprawdźmy teraz w ramach przykładu działanie opcji ON DELETE CASCADE. W tym celu usuniemy klucz obcy z tabeli Rachunki, a następnie odtworzymy go z opcją ON DELETE CASCADE. Po wykonaniu tej operacji spróbujmy usunąć jeden wiersz z tabeli Klienci. Niech będzie to klient o identyfikatorze 2, który z tego co pamiętamy posiada dwa Rachunki (dwa wiersze w tabeli Rachunki z jego identyfikatorem w kolumnie klucza obcego). Po wykonaniu tego polecenia i pobraniu pełnej listy rachunków okazuje się, że zgodnie z naszymi oczekiwaniami, po usunięciu klienta usunięte zostały jego rachunki. W tym miejscu jeszcze raz warto podkreślić, że jest to opcja bardzo niebezpieczna i przy bardziej złożonej strukturze tabel polecenie usunięcia jednego wiersza może spowodować „czystkę" w bazie danych.

Warto wspomnieć o jeszcze kilku cechach kluczy obcych. Po pierwsze klucz obcy nie musi być pojedynczą kolumną. Można go zdefiniować na kilku kolumnach, lecz trzeba wtedy pamiętać, że jeżeli te kolumny dopuszczają wartość null, a którakolwiek z nich będzie miała tę wartość, to pozostałe nie będą sprawdzane pod kątem zgodności z regułą klucza.

Klucz obcy nie musi odwoływać się do kolumny lub kolumn z innej tabeli. W pewnych przypadkach tworzy się klucze obce odwołujące się do kolumn w tej samej tabeli. Mówimy wtedy o samozłączeniu. Takie rozwiązanie bywa stosowane na przykład do budowania danych o strukturze hierarchicznej. Przykładowo weźmy tabelę pracownicy, która ma kolumny:

- PracownikID (klucz podstawowy)
- Nazwisko
- Imie
- kierownikID (klucz obcy, dopuszczalna wartość null)

Tak prosta struktura tabeli pozwala na zdefiniowanie hierarchii pracowników poprzez określenie, kto jest czyim kierownikiem za pomocą wartości kolumny KierownikID, która jest kluczem obcym odwołującym się do kolumny z tej samej tabeli. W takiej hierarchii „szef wszystkich szefów" będzie miał wartość null w kolumnie kierownik id. Rozwiązania oparte o taki schemat są bardzo łatwe w implementacji, lecz nieco uciążliwe w obsłudze, szczególnie gdy próbujemy się poruszać po hierarchii w różnych kierunkach

Alternatywą dla takich rozwiązań jest typ danych hierarchyID lub XML. Mają one unikalne cechy, które predestynują je do stosowania w określonych scenariuszach. Warto o nich poczytać zanim zdecydujemy się na zastosowanie konkretnego pomysłu.

4 TRANSAKCJE

Bardzo istotnym zagadnieniem, związanym z relacyjnymi bazami danych, jest **mechanizm transakcji**. Umożliwia on korzystanie z takich baz danych w ramach „poważnych" systemów informatycznych, które nie mogą sobie pozwolić na pojawianie się stanów nieustalonych w ramach danych.

Przy tworzeniu aplikacji bazodanowych rzadko zastanawiamy się, czy baza zapewnia stabilność i bezpieczeństwo danych. Przyjmujemy, że tak. Warto natomiast zdawać sobie sprawę, jakie mechanizmy leżą u podstaw tego przekonania. W ramach wykładu 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.

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.

Skoro istnieje problematyka spójności danych, 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 umożliwia równoległe wykonywanie niektórych operacji modyfikacji danych i ich odczytu z poziomu innych transakcji.

Domyślnie SQL Server jest skonfigurowany w ten sposób, że transakcje obsługuje w trybie AUTOCOMMIT. Oznacza to, że transakcje są 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).

Transakcje mogą być zagnieżdżane. Może się to odbywać nie tylko w sposób pokazany na slajdzie, ale także w bardziej złożony – na przykład mamy rozpoczętą transakcję, a w jej ramach wywołujemy procedurę składowaną, która wewnątrz swojego kodu zawiera także transakcję. Procedura może próbować wykonać jakąś operację w ramach swojej transakcji, a w przypadku porażki wycofać ją, wykonać alternatywny kod i zakończyć działanie. W takiej sytuacji zewnętrzna transakcja może być kontynuowana i zatwierdzona poprawnie. 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 się zapoznać z mechanizmem działania transakcji zagnieżdżonych i przećwiczyć go w praktyce.

SQL Server zawiera wbudowaną zmienną @@TRANCOUNT, w której jest przechowywany aktualny poziom zagnieżdżenia. Jeśli nie ma w danej chwili żadnej aktywnej transakcji, to 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 złożony 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. Umożliwia to budowanie alternatywnych ścieżek realizacji transakcji.

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 do poprzedniej. 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.

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 eliminu-

je 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

Rysunek 3.
Występowanie anomalii w różnych poziomach izolacji transakcji

Przy realizacji wielu transakcji jednocześnie, czasem dochodzi do jeszcze jednego 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. Dzięki temu druga transakcja uzyskuje wszystkie potrzebne zasoby i realizuje wszystkie zaplanowane czynności.

Zakleszczenie 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 umożliwia 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 umożliwiają 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 umożliwia uzyskanie lepszej wydajności bazy danych.

5 WYZWALACZE

Kolejnym ciekawym mechanizmem stosowanym w bazach danych są **wyzwalacze** (ang. *triggers*). Wyzwalacze można porównać do procedur obsługi zdarzeń w programowaniu obiektowym. W ramach wykładu zapoznamy się z poszczególnymi rodzajami wyzwalaczy oraz z typowymi scenariuszami korzystania z nich. Wyzwalacze, działające dla zdarzeń związanych z poleceniami **języka DML**, można podzielić na:

- wyzwalacze AFTER – kod w nich zawarty jest wykonywany po operacji uruchamiającej wyzwalacz, ale co istotne w ramach tej samej transakcji;
- wyzwalacze INSTEAD OF – ich kod jest wykonywany zamiast operacji uruchamiającej wyzwalacz; właściwa operacja nie dochodzi do skutku.

Dla jednego zdarzenia na jednej tabeli można tworzyć wiele wyzwalaczy. W takiej sytuacji ważne jest, aby pamiętać, że kolejność ich wykonania jest nieokreślona. Jedyne co możemy w tej sprawie zrobić, to określić który z nich ma się wykonać jako pierwszy, a który jako ostatni. Realizuje się to za pomocą procedury składowanej sp_settriggerorder.

SQL Server udostępnia dla kodu wyzwalaczy dwie specjalne tabele. Są to tabele **inserted** i **deleted**. Są one w pełni obsługiwane przez serwer i utrzymywane w pamięci, więc dostęp do nich jest szybki. Gdy rozpoczyna się wykonanie kodu wyzwalacza, tabele te są napełniane odpowiednimi danymi. W przypadku wyzwalacza skojarzonego z poleceniem INSERT – tabela **inserted** będzie zawierała dodawane wiersze. W przypadku wyzwalacza skojarzonego z poleceniem DELETE – tabela **deleted** będzie zawierała usuwane dane. W przypadku wyzwalacza skojarzonego z poleceniem UPDATE – tabela **deleted** będzie zawierała oryginalne wartości modyfikowanych danych, a tabela **inserted** ich nowe wartości.

Istotną cechą wyzwalaczy jest fakt, że mogą one być uruchomione nie tylko dla operacji modyfikującej pojedynczy wiersz. Wyzwalacz jest uruchamiany w odpowiedzi na wykonanie jednego polecenia, ale to polecenie może na przykład dodać lub usunąć kilkaset wierszy jednocześnie. Dlatego właśnie budując kod wyzwalacza powinniśmy uwzględniać, że może on wykonywać się dla wielu wierszy modyfikowanych jednym poleceniem uruchamiającym wyzwalacz.

Osobną grupą wyzwalaczy są **wyzwalacze DDL**. Jak sama nazwa wskazuje reagują one na zdarzenia związane z poleceniami **języka DDL** (ang *Data Definition Language*). Takimi poleceniami są: CREATE, ALTER, DROP, GRANT, DENY, REVOKE, UPDATE STATISTICS. Przeznaczenie wyzwalaczy DDL jest nieco inne niż wcześniej omawianych wyzwalaczy DML. Podstawową rolą wyzwalaczy DDL jest wspomaganie mechanizmów audytu i śledzenia zmian w strukturze bazy danych oraz ograniczanie możliwości manipulowania tą strukturą. W kodzie wyzwalacza DDL dostępna jest specjalna funkcja EVENTDATA(), która zwraca szczegółowe informacje o zdarzeniu w formie XML (zwraca wartość typu XML). Wyzwalacze DDL można tworzyć na poziomie bazy danych lub całego serwera. Typowe zastosowanie wyzwalaczy to:

- Implementacja złożonych reguł integralności.
- Rejestrowanie działań użytkowników.
- Zabezpieczenie przed niepożądanymi operacjami.
- Wyliczanie wartości liczbowych na podstawie zmian w tabelach.

W każdym przypadku zastosowania wyzwalaczy jest to decyzja wynikająca z chęci automatyzacji pewnych działań w zależności od wystąpienia odpowiedniego zdarzenia.

6 INDEKSY

Zanim zaczniemy zgłębiać tematykę indeksów oraz ich wpływy na wydajność, wypadałoby zapoznać się choćby w niewielkim stopniu z mechanizmami leżącymi u podstaw działania SQL Servera. Jednym z istotnych zagadnień jest tu sposób, w jaki dane są fizycznie przechowywane w bazie danych.

6.1 FIZYCZNA ORGANIZACJA DANYCH W SQL SERVER 2008

Gdy myślimy o tabeli, to wizualizujemy sobie coś na kształt zbioru wierszy składających się z kolumn zawierających dane różnego typu:

Nie zastanawiamy się, jak te dane są przechowywane fizycznie na dysku, ani jaki wpływ na wydajność mogą mieć nasze decyzje podjęte przy projektowaniu tabeli. Warto jednak zadać sobie nieco trudu i zapoznać

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

Rysunek 4.
Przykładowa zawartość tabeli

się z fizycznym sposobem przechowywania danych w bazie. Zrozumienie podstaw pozwoli później zrozumieć dlaczego w takiej czy innej sytuacji wykonanie zapytania czy modyfikacji danych trwa długo.

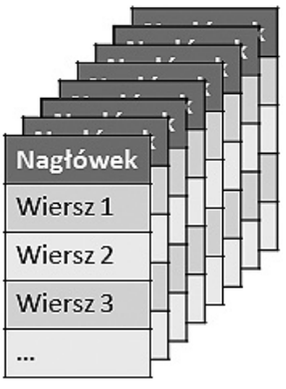
6.2 STRONY I OBSZARY

Najmniejszą jednostką przechowywania danych w SQL Server jest **strona** (ang. *page*). Jest to blok o wielkości 8KB, składający się z nagłówka i 8060 bajtów na dane z wiersza (lub wierszy). Przy założeniu, że wiersz tabeli musi się zmieścić na stronie jasno widać, że maksymalny rozmiar wiersza to 8060 bajtów. Trochę mało? Niekoniecznie. Część danych o rozmiarze przekraczającym 8KB jest zapisywana na innych stronach, a w samym wierszu umieszczany jest tylko wskaźnik do pierwszej z tych stron. SQL Server rozróżnia 9 rodzajów stron przechowujących informacje o rozmaitym znaczeniu:

- Strony danych (data) zawierają wszystkie dane z wiersza, z wyjątkiem kolumn typów: text, ntext, image, nvarchar(max), varchar(max), varbinary(max), xml.
- Jeżeli wiersz nie mieści się w limicie długości 8060 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 GAM, SGAM i IAM – wrócimy do nich w dalszej części wykładu, gdy poznamy kolejne pojęcia dotyczące fizycznego przechowywania danych.

Wymieniliśmy tylko 6 rodzajów, żeby niepotrzebnie nie komplikować dalszych rozważań. Dla uporządkowania warto wspomnieć o pozostałych trzech: Page Free Space, Bulk Changed Map, Differential Changed Map. Pierwsza zawiera informacje o zaalokowanych stronach i wolnym miejscu na nich. Pozostałe dwa rodzaje są wykorzystywane do oznaczania danych zmodyfikowanych w ramach operacji typu bulk oraz do oznaczania zmian od ostatnio wykonanej kopii zapasowej.

Podstawową jednostką alokacji w SQL Server nie jest jednak strona, tylko zbiór ośmiu stron zwany **obszarem** (ang. *extent*).

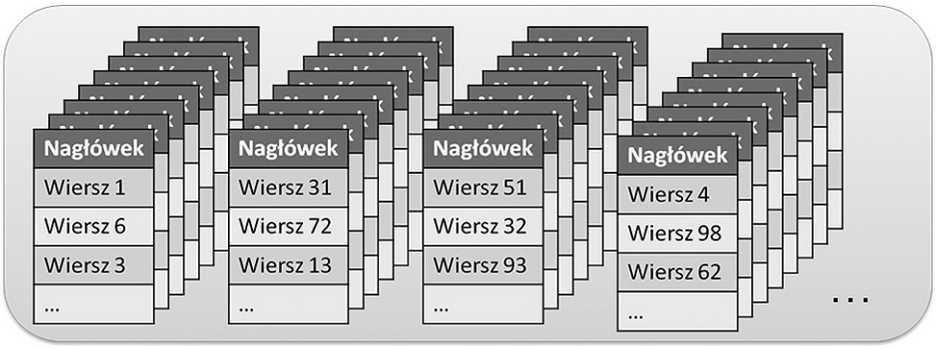


Rysunek 5.
Strony danych

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.

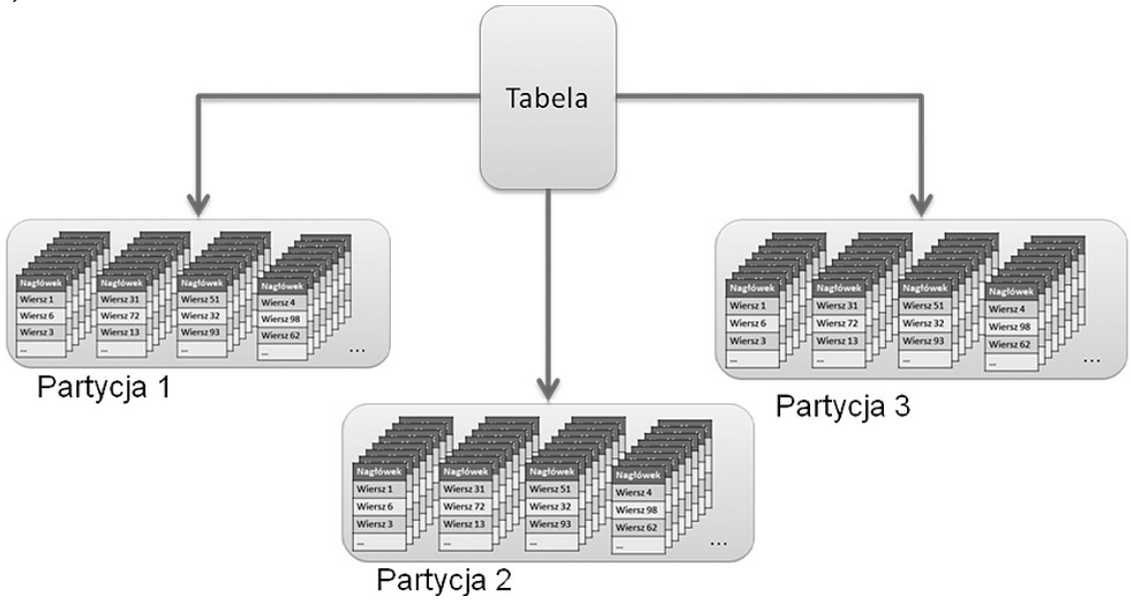
6.3 STERTY

Jeżeli tabela nie zawiera żadnego indeksu, 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 zawierające dane klientów o nazwisku „Kowalski". Stertę można wyobrazić sobie jak na rys. 6.



Rysunek 6.
Sterta

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



Rysunek 7.
Organizacja sterty podzielona na partycje

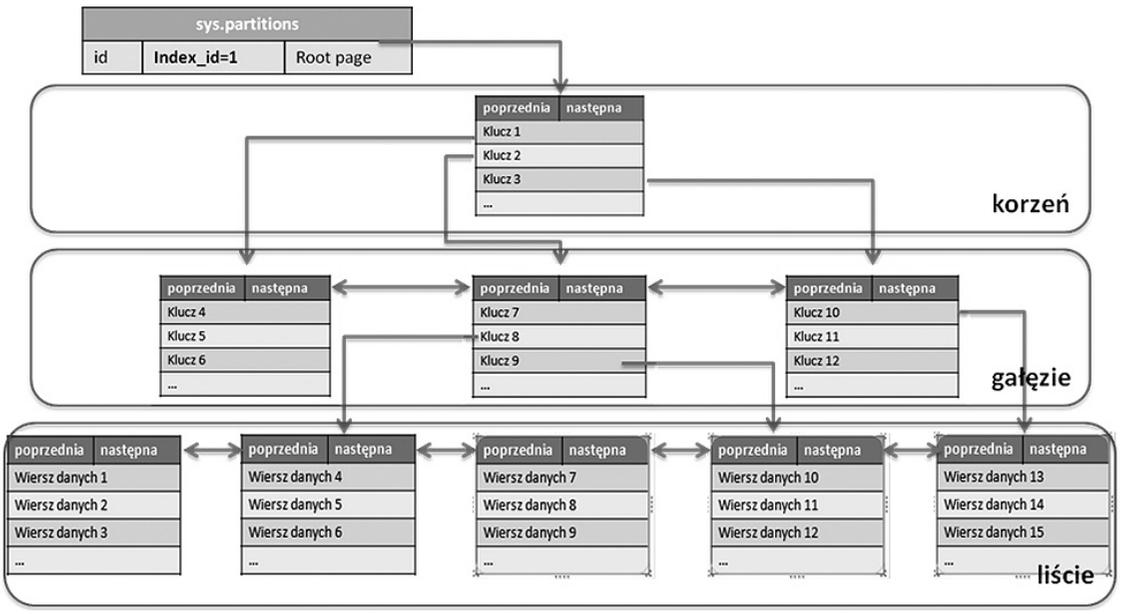
Gdy SQL Server alokuje miejsce w plikach bazy danych, wypełnia je obszarami, które wstępnie są oznaczone jako wolne. Podobnie wszystkie strony w obszarach są oznaczone jako puste. W jaki sposób są przechowywane informacje na temat tego, czy dany obszar lub strona są wolne lub należą do jakiegoś obiektu? 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. Kluczem do uzyskania dostępu do danych z tabeli jest możliwość dostania się do strony IAM tej tabeli. Infor-

macje na temat lokalizacji stron IAM dla poszczególnych obiektów znajdują się we wpisach w tabelach systemowych.

6.4 INDEKSY ZGRUPOWANE I NIEZGRUPOWANE

Poznaliśmy już z grubsza sposób przechowywania danych w tabeli, dla której nie stworzono indeksów. Cechą charakterystyczną był fakt nieuporządkowania stron i wierszy należących do jednej tabeli, co wymuszało przy każdej operacji wyszukiwania danych w tabeli przeszukanie wszystkich wierszy. 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 jakieś inne mechanizmy umożliwiające efektywne wyszukiwanie. Rzeczywiście takowe istnieją. Są to indeksy, występujące w dwóch podstawowych wariantach, jako **indeksy: zgrupowane** (ang. *clustered*) i **indeksy niezgrupowane** (ang. *nonclustered*).

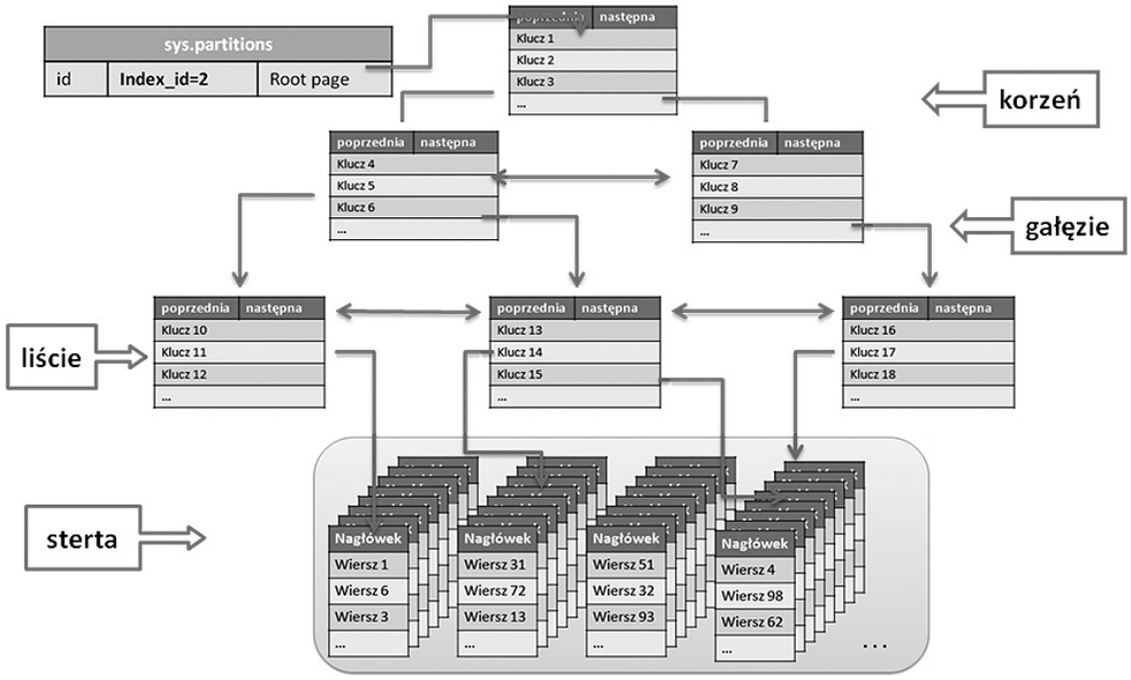
Indeks zgrupowany ma postać drzewa zrównoważonego (B-tree). Na poziomie korzenia i gałęzi znajdują się strony indeksu zawierające kolejne wartości klucza indeksu uporządkowane rosnąco. Na poziomie liści znajdują się podobnie uporządkowane strony z danymi tabeli. To właśnie jest cechą charakterystyczną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



Rysunek 8.
Struktura indeksu zgrupowanego

Specyfika indeksu zgrupowanego polega na fizycznym porządkowaniu danych w tabeli według wartości klucza indeksu. W związku z tym ten indeks 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.

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. To z kolei przenosi się na zmniejszenie ilości stron całości indeksu. Daje to w efekcie mniej operacji wejścia/wyjścia wykonywanych 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.



Rysunek 9.
Struktura indeksu niezgrupowanego dla sterty

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.

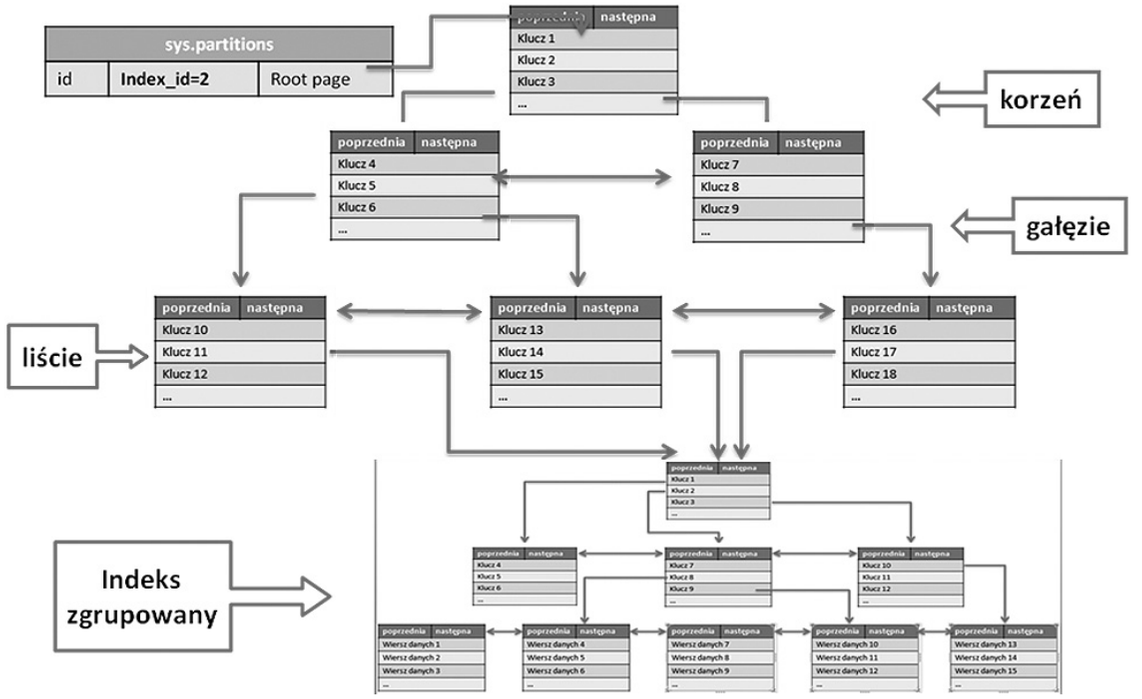
Indeksy niezgrupowane mają strukturę zbliżoną do zgrupowanych. Zasadnicza różnica polega na zawartości liści indeksu. O ile indeksy zgrupowane mają w tym miejscu strony danych, to indeksy niezgrupowane – strony indeksu. Strony te zależnie od wariantu indeksu niezgrupowanego zawierają oprócz klucza różne 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 mającej 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).

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

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. Dysponuje rozbudowanym buforem 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.

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) są dodawane dodatkowe kolumny. Jeżeli liście indeksu niezgrupowanego zawierają wszystkie kolumny zwracane



Rysunek 10.
Struktura indeksu niezgrupowanego dla tabeli z indeksem zgrupowanym

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).

6.5 INDEKSY POKRYWAJĄCE

Żeby zademonstrować sposób działania indeksów pokrywających załóżmy następującą sytuację. W bazie istnieje tabela zawierająca dane klientów. W jej skład wchodzi kilka kolumn: ID, Nazwisko, Imie, Email, DataOstatniegoZamowienia. Na tabeli został stworzony indeks zgrupowany na kolumnie ID, oraz indeks niezgrupowany na kolumnie Nazwisko. Jeżeli w takim przypadku realizowane będzie zapytanie, które co prawda w klauzuli WHERE zawiera warunek tylko dla kolumny nazwisko (zawartej w indeksie niezgrupowanym), ale na liście kolumn wyjściowych zawiera także inne kolumny (w naszym przypadku kolumna Email), to indeks niezgrupowany nie zostanie wykorzystany, gdyż wartości kolumn spoza indeksu muszą zostać pobrane ze stron danych. Zapytanie zostanie zrealizowane poprzez skanowanie indeksu zgrupowanego. Jeżeli usuniemy z listy kolumn wyjściowych kolumnę Email i wykonamy zapytanie ponownie, to tym razem indeks niezgrupowany okaże się przydatny i zostanie na nim wykonana operacja wyszukiwania w indeksie. Będzie ona mniej kosztowna od skanowania indeksu zgrupowanego, gdyż nie wymaga dostępu do stron danych. Żeby osiągnąć ten sam efekt z kolumną email na liście wyjściowej należy dodać ją do indeksu niezgrupowanego (jako część klucza lub nie). Po takiej modyfikacji osiągniemy założony cel – zapytanie zostanie zrealizowane z wykorzystaniem operacji wyszukiwania w indeksie niezgrupowanym.

Mechanizm indeksów pokrywających wygląda atrakcyjnie 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.

6.6 PLANY WYKONANIA ZAPYTANIA

Gdy zlecamy 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 na wypadek, gdyby mógł 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 aż do chwili 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 (Actual Execution Plan) 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 do 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 wykładu.

Wśród całej gamy informacji wyświetlanych w szczegółach wybranego operatora, dla nas najistotniejsze są te związane z kosztem wykonania danego etapu. W dalszych przykładach będziemy się na nich opierać prezentując zmiany kosztu wykonania zapytania w zależności od podjętych kroków przy optymalizacji zapytania.

6.7 STATYSTYKI

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 nosi nazwisko Kowalski, a jedynie dwóch Nowak. Na podstawie samego faktu istnienia indeksu na kolumnie Nazwisko trudno ocenić czy sensownie jest go wykorzystać przy wyszukiwaniu Kowalskich lub Nowaków. Po przejrzeniu statystyk może okazać się, że dla Kowalskiego nie ma co zaprzątać sobie głowy indeksami, natomiast w przypadku Nowaka 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 5 minut. Skutki mogą być opłakane. Z tego powodu, jeżeli mamy do czynienia z sytuacja, 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 nam oszczędzić sporo czasu.

7 PROCEDURY I FUNKCJE SKŁADOWANE

Kolejnym elementem dostępnym w ramach bazy danych, który wzbogaca logikę bazy danych o nawet bardzo złożone mechanizmy, są procedury składowane oraz funkcje użytkownika. Służą one do grupowania kodu SQL w większe bloki, które realizują konkretne zadanie w ramach bazy danych. W ramach wykładu dokonamy krótkiego przeglądu możliwości oferowanych przez procedury składowane i funkcje użytkownika.

Procedury składowane to bloki kodu (poleceń) SQL, które można wykonywać jako całość. Procedury składowane przypominają funkcje czy metody znane z języków programowania. Można przy ich wywołaniu przekazywać parametry wejściowe i wyjściowe, a także odbierać zwracany przez nie kod powrotu (liczba całkowita). Korzystanie z procedur składowanych ma wiele zalet. Przede wszystkim umożliwia grupowanie poleceń SQL, których wykonanie jest potrzebne do realizacji operacji biznesowej. W połączeniu z możliwością nadawania uprawnień do wykonania procedury, umożliwia to łatwe budowanie systemu uprawnień do realizacji określonych operacji biznesowych. Dodatkowym aspektem jest fakt, że budowanie logiki aplikacji w oparciu o wywołania procedur składowanych umożliwia odcięcie się od szczegółów implementacyjnych bazy danych. Aplikacja (lub użytkownik) nie musi np. wiedzieć, że dane klienta są przechowywane w trzech tabelach. Wystarczy, że przekaże je jako parametry wejściowe – resztę załatwi kod zawarty w procedurze.

Funkcje użytkownika – **UDF** (ang. *User Defined Functions*) są tworem zbliżonym do procedur składowanych. Podstawową różnica jest to, że można ich wywołania wykorzystywać w wyrażeniach i zapytaniach w miejscach wartości które zwracają. Funkcje użytkownika mogą zwracać wartości skalarne lub tabelaryczne. W przypadku wartości tabelarycznych funkcja może być zbudowana w oparciu o pojedyncze zapytanie SELECT lub o wiele wyrażeń prowadzących do wygenerowania wynikowego zbioru wierszy.

8 KOPIE BEZPIECZEŃSTWA BAZ DANYCH

Dla zapewnienia bezpieczeństwa danych istotne jest wykonywanie ich kopii zapasowych. SQL Server zawiera zestaw poleceń i narzędzi, które służą do tego celu. Mimo możliwości oferowanych przez narzędzia, największa odpowiedzialność leży po stronie użytkownika. Nawet najlepsze narzędzia nie zagwarantują sprawnego odtworzenia stanu systemu po awarii, jeżeli wcześniej zabrakło planu zarówno wykonywania kopii zapasowych, jak i odtwarzania stanu systemu po awarii.

Pierwszym zagadnieniem, które trzeba rozważyć przy planowaniu strategii wykonywania kopii zapasowych bazy danych jest **model odtwarzania** (ang. *recovery model*). Od niego zależy, jakie rodzaje kopii zapasowych będziemy mogli wykonywać oraz w jakim stopniu odtwarzać system po awarii.

Najprostszym trybem odtwarzania jest tryb Simple Recovery. W tym trybie osoba administrująca bazą danych jest zwolniona z obowiązków związanych z utrzymaniem logu transakcji (kopie zapasowe, obcinanie i kontrolowanie rozmiaru). Kosztem jest natomiast możliwość odtworzenia stanu systemu po awarii jedynie do stanu z ostatniej kopii zapasowej (pełnej lub pełnej i różnicowej).

Tryb Full Recovery daje pełne możliwości w zakresie wykonywania i odtwarzania kopii zapasowych. Oprócz kopii pełnych i różnicowych można wykonywać kopie logu transakcji. Daje to znacznie większe możliwości przy odtwarzaniu stanu systemu po awarii. Z drugiej strony administrator jest obarczony dodatkowymi obowiązkami wynikającymi z konieczności dbania o log transakcji.

Do wykonywania kopii zapasowych służy polecenie BACKUP. Za jego pomocą można wykonywać kopie całej bazy (BACKUP DATABASE), logu transakcji (BACKUP LOG) lub poszczególnych plików wchodzących w skład bazy danych (BACKUP FILE).

Kopia zapasowa całej bazy danych może być wykonana jako **kopia pełna** (ang. *full backup*) lub **kopia różnicowa** (ang. *differential backup*). Wykonywanie kopii zapasowej logu transakcji jest możliwe tylko przy bazie działającej w trybie full recovery. Wykonywanie kopii zapasowych nie jest celem samym w sobie. Jego istotą jest umożliwienie odtworzenia stanu systemu sprzed awarii. Polecenie, które służy do odtwarzania danych (RESTORE) z kopii zapasowej posiada takie same warianty jak polecenie BACKUP (DATABASE, LOG, FILE).

Przy odtwarzaniu danych z więcej niż jednej kopii zapasowej trzeba pamiętać, by wszystkie kopie z wyjątkiem ostatniej odtwarzane były z opcją NORECOVERY, która pozostawia bazę w stanie niespójności i umożliwia odtwarzanie danych z kolejnych kopii. Dopiero ostatnia kopia jest odtwarzana z opcją RECOVERY (jest to domyślne działanie polecenia RESTORE) co powoduje wycofanie niezatwierdzonych transakcji i powrót bazy danych do stanu stabilnego .

Sama umiejętność korzystania z poleceń czy narzędzi do wykonywania kopii zapasowych i odtwarzania z nich baz danych nie wystarczy, żeby móc powiedzieć, że nasze dane są bezpieczne. Istotniejszym elementem jest właściwie opracowana i dopasowana do wymagań strategia wykonywania kopii zapasowych i odtwarzania danych po awarii. Zaplanowanie takiej strategii nie jest bynajmniej zadaniem trywialnym i często zdarza się, że po awarii (mimo posiadania całego zestawu kopii zapasowych) nie udaje się odtworzyć danych i doprowadzić ich do wymaganego stanu.

Podstawą planowania strategii są wymagania dotyczące bezpieczeństwa danych:

- Jak długo może trwać proces przywracania bazy danych do stanu używalności?
- Na utratę ilu danych można sobie pozwolić (z ostatniej godziny, trzech godzin, kwadransa, czy najlepiej bez strat danych).

Duży wpływ na postać strategii ma rozmiar bazy danych. Im większy, tym trudniej osiągnąć zgodność z wymaganiami.

Integralnym składnikiem strategii jest plan (procedura) odtwarzania bazy po awarii. Bez takiego dokumentu bardzo łatwo doprowadzić do sytuacji, w której nerwy i stres towarzyszące awarii spowodują, że administrator popełni błąd, który może zniweczyć szanse na odtworzenie danych do wymaganej postaci. Nie bez kozery administratorzy mawiają, że jeżeli dojdzie do awarii to pierwsza rzecz, którą należy wykonać, to usiąść z dala od klawiatury i uspokoić się. Kolejnym krokiem jest zwykle wykonanie kopii zapasowej logu transakcji. Dopiero potem przechodzi się do dalszych kroków. Jeżeli są one precyzyjnie opisane i najlepiej przećwiczone praktycznie, to szanse na odzyskanie danych znacząco rosną.

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

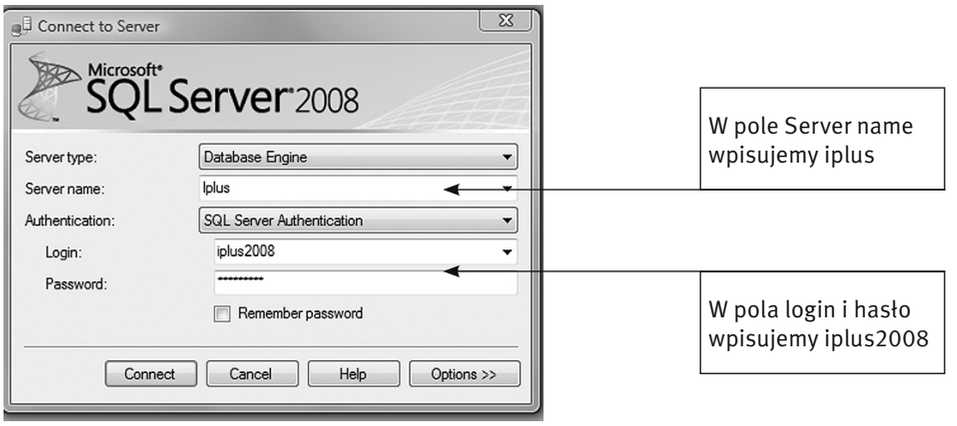
WARSZTATY

W ramach warsztatów będzie wykorzystywany System Zarządzania Bazami Danych MS SQL Server 2008 Express. Poniżej zamieszczona jest instrukcja, jak ściągnąć z Internetu i zainstalować to oprogramowanie.

Ćwiczenie 1. **Zapoznanie się ze środowiskiem MS SQL Server 2008**

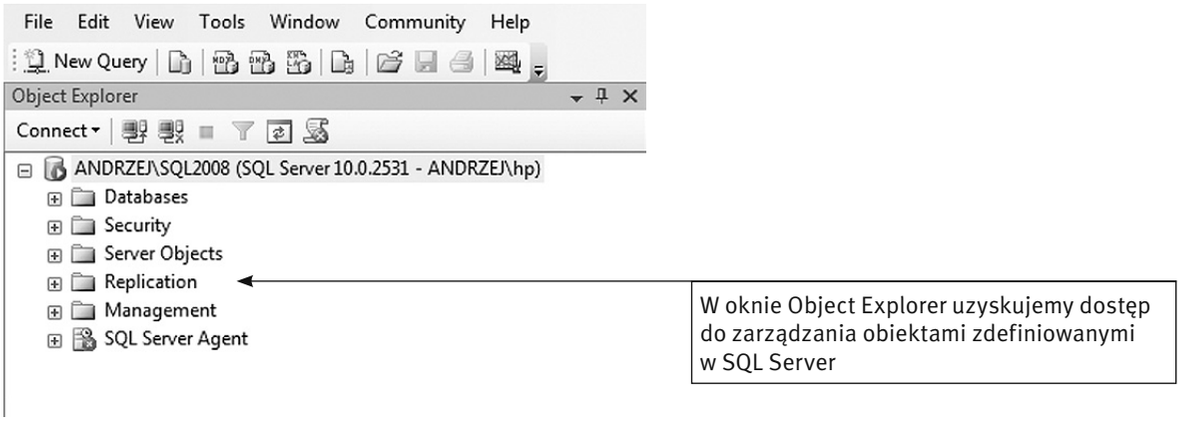
Wspólnie z prowadzącym warsztaty rozpoznajemy środowisko SZBD MS SQL Server 2008. Korzystanie z tego systemu umożliwia specjalne oprogramowanie SQL Server Menagement Studio.

1. Uruchamiamy SQL Server Menagement Studio (lokalizację programu poda prowadzący warsztaty).
2. Po uruchomieniu programu przechodzimy do logowania się do SQL Servera – na ekranie pojawi się okienko do logowania (rys. 11), w którym wpisujemy w pola wartości takie, jak podane na rysunku.



Rysunke 11.
Logowanie do SQL Server 2008

3. Po poprawnym wpisaniu powyższych wartości uruchomione zostanie oprogramowanie i pojawia się okna SQL Server Menagement Studio (rys. 12).



Rysunek 12.
Okna SQL Server Menagement Studio

4. Wspólnie z prowadzącym warsztaty poznajemy wybrane elementy środowiska SQL Servera.

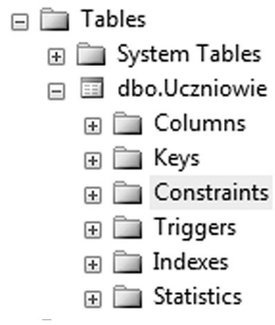
Ćwiczenie 2. **Definiowanie reguły poprawności**

W tabeli zawierającej kolumnę o nazwie Pesel chcemy zdefiniować regułę zapewniającą, że zapisana tam wartość będzie zawierała dokładnie 11 cyfr. Zadanie realizujemy według następującego schematu:

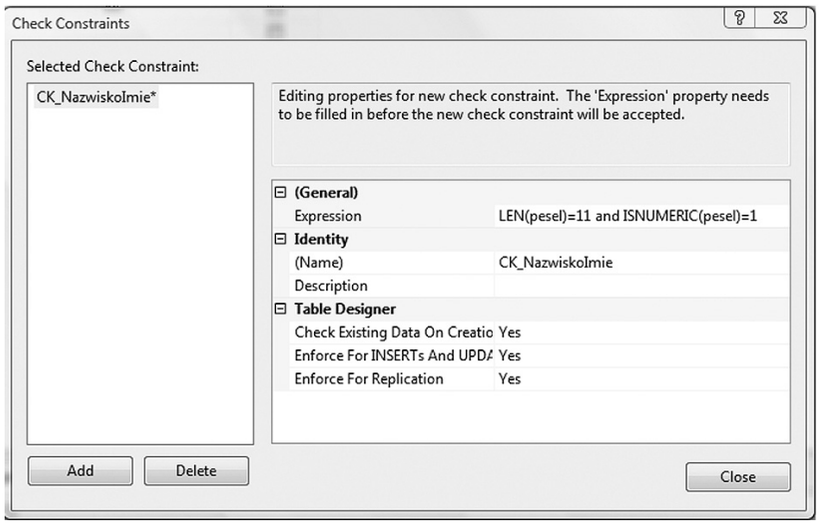
1. Tworzymy nową tabelę (każdy uczeń nazywa tabelę swoim imieniem i nazwiskiem) w bazie danych InfoPlusTesty o następującej strukturze:

Column Name	Data Type	Allow Nulls
idosoby	int	☐
Nazwisko	varchar(50)	☐
Imie	varchar(50)	☐
Pesel	varchar(11)	☐
idmiasta	int	☐

2. Rozwijamy folder Tables.
3. Wybieramy własną, zdefiniowana w poprzednim punkcie tabelę.
4. Rozwijamy folder tej tabeli – poniżej przykładowa zawartość takiego folderu.



5. Wybieramy folder Constraints i po kliknięciu prawym klawiszem myszy wybieramy opcję New Constraint – pojawia się okienko definiowania reguły (rys. 13). :



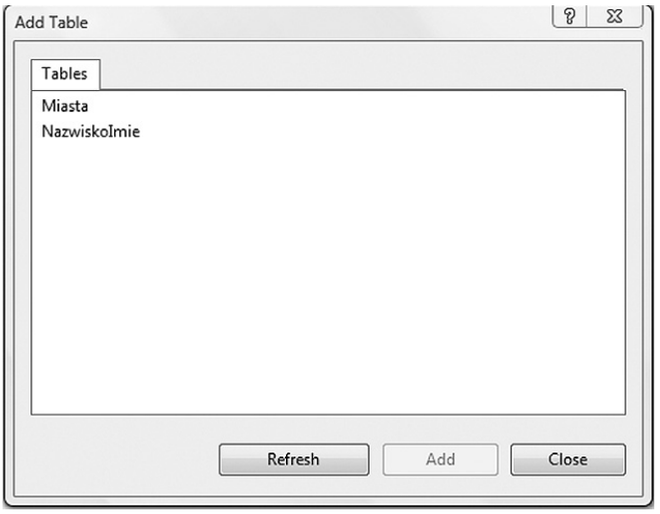
Rysunek 13.
Okno definiowania reguł

6. W polu Expression wpisujemy wyrażenie logiczne według następującej formuły :
   LEN(pesel)=11 and ISNUMERIC(pesel)=1.
7. Po zapisaniu wyrażenia naciskamy przycisk Close.
8. Po poprawnym wykonaniu zadania – określona reguła zaczyna działać.
   Sprawdzić działanie reguły poprzez próby wprowadzania danych, które tej reguły nie spełniają.

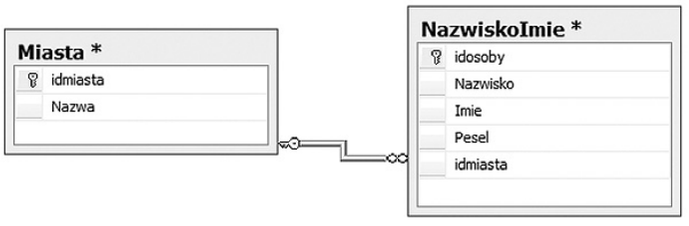
Ćwiczenie 3. **Definiowanie reguł integralności referencyjnej**

W ramach tego ćwiczenia zdefiniujemy regułę integralności referencyjnej oraz utworzymy diagram bazy danych. Zadanie realizujemy według następującego schematu :

1. Wybieramy bazę danych o nazwie ***IntoPlusTest***.
2. Rozwijamy folder tej bazy danych.
3. Wybieramy folder Database Diagrams i po kliknięciu prawym klawiszem myszy wybieramy opcję New Database Diagram – pojawi się okno z listą tabel zdefiniowanych w bazie danych(w rzeczywistości, oprócz tabeli o nazwie Miasta będą się znajdowały tabele definiowane przez innych uczestników:



4. Wybieramy tabelę definiowaną przez siebie oraz tabele Miasta i naciskamy przycisk Add – pojawi się obszar diagramu.
5. Widoczne tabele są zdefiniowane w bazie danych bez określenia powiązań pomiędzy kluczami obcymi i podstawowymi.
6. W ramach definiowania diagramu należy przeciągnąć myszą klucz podstawowy do odpowiadającego mu klucza obcego – po wykonaniu takiej operacji na diagramie pojawi się powiązanie. Po wykonaniu zadania diagram powinien mieć postać, jak poniżej:



7. Po wykonaniu zadania zamykamy okno diagramu i nadajemy mu nazwę (nazwisko i imię wykonującego).
8. Poprawne zapisanie dokonanych zmian powoduje uruchomienie mechanizmu więzów integralności referencyjnej.
9. Sprawdzamy działanie mechanizmu poprzez próby wykonania niedozwolonych operacji.
10. Wyjaśniamy pytania i wątpliwości z prowadzącym warsztaty.

Ćwiczenie 4. **Prezentacja działania mechanizmu transakcyjnego**

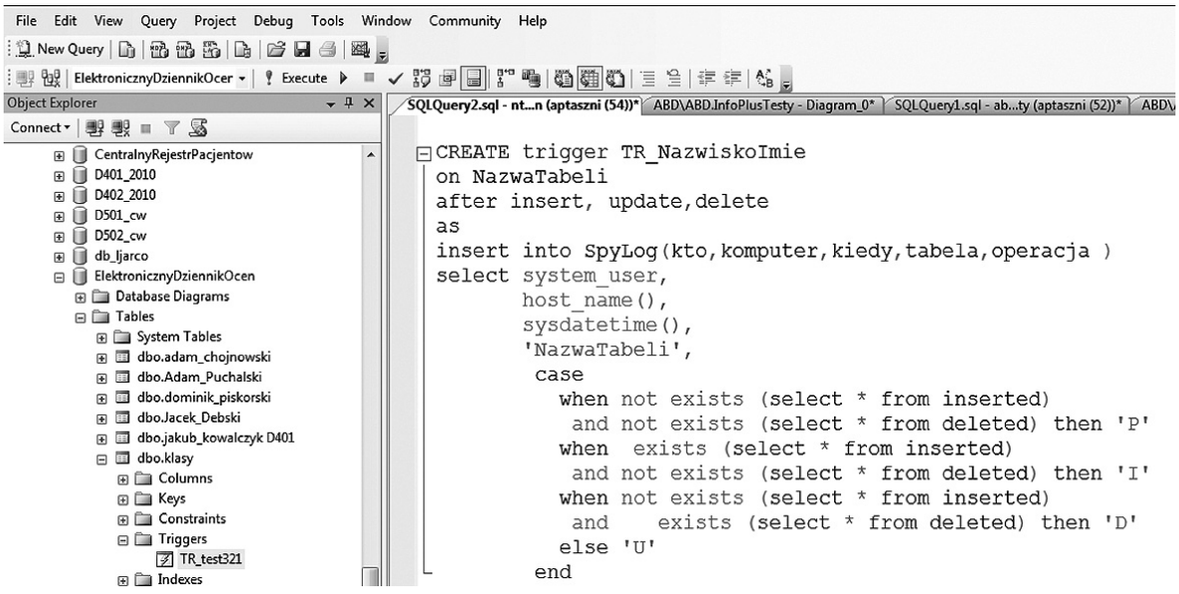
W ramach tego ćwiczenia, prowadzący zademonstruje działanie mechanizmu transakcyjnego. Zaprezentowane zostanie definiowanie transakcji oraz działanie różnych poziomów izolacji transakcji oraz zjawisko zakleszczenia.

Ćwiczenie 5. **Programowanie wyzwalacza**

W bazie danych InfoPlusTest jest tabela o nazwie InspekcjaDzialania o następującej strukturze;

Tabela InspekcjaDzialania ma za zadanie rejestrować operacje zmiany danych realizowane w tabelach bazy danych InfoPlusTest. Każdy uczestnik zdefiniuje wyzwalacz, dla swojej tabeli definiowanej w ćwiczeniu 2, który będzie zapisywał informacje o wykonywanych modyfikacjach danych. Zadanie realizujemy według następującego schematu:

1. Klikamy na przycisku New Query (nowe zapytanie) – uruchamia się okienko edycyjne, w którym będziemy wpisywać kod definiujący wyzwalacz;

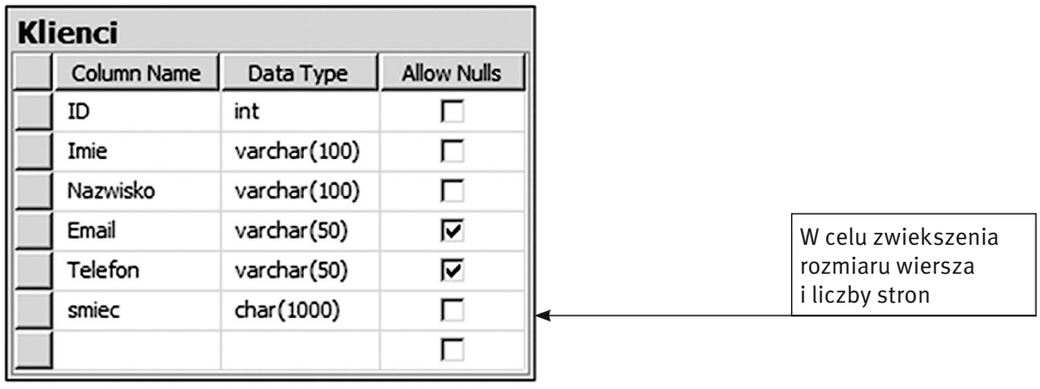


Rysunek 14.
Definiowanie wyzwalacza

2. Wykonując zapisane polecenie klikamy na przycisku (poleceniu) Execute (lub naciskamy klawisz F5), patrz rys. 14.
3. Po poprawnym wykonaniu polecenia zdefiniowany wyzwalacz powinien być gotowy do działania.
4. Sprawdzamy działanie wyzwalacza wykonując operacje zmiany danych dla swojej tabeli i przeglądamy zapisy realizowane w tabeli InspekcjaDzialania.
5. W kodzie wyzwalacza pokazanym na rys.14, każdy uczestnik modyfikuje te miejsca kodu, w których występuje słowo „NazwiskoImie", zastępując je swoim imieniem i nazwiskiem oraz słowo „NazwaTabeli" wpisując nazwę tabeli definiowanej w ramach ćwiczenia 2.

Ćwiczenie 6. **Optymalizacja przykładowego zapytania**

W ramach tego ćwiczenia, prowadzący zademonstruje działanie indeksów i wpływ indeksów na generowane plany wykonania zapytań. Zapytania będą dotyczyły tabeli Klienci, w której na początku nie ma żadnego indeksu.

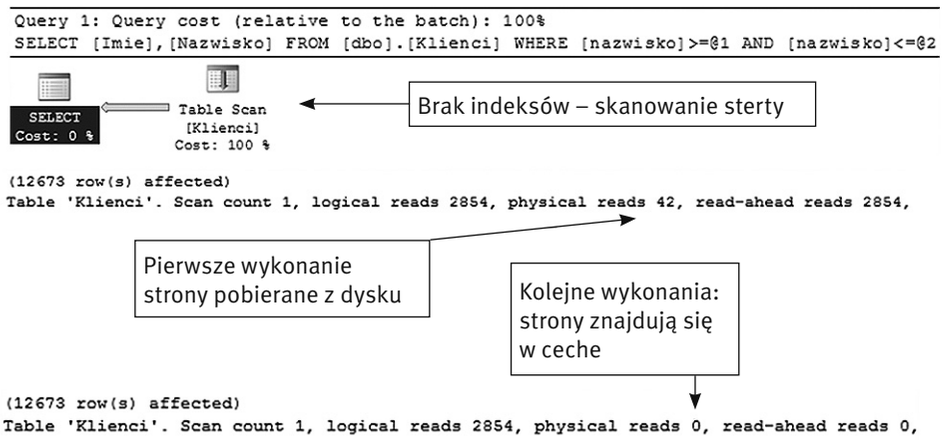


Rysunek 15.
Struktura tabeli Klienci

Z tego powodu można przewidywać, że operacją wykorzystaną do realizacji zapytania będzie skanowanie tabeli. Przykładowe zapytanie ma postać:

```
SELECT
  Imie
 ,Nazwisko
FROM
  dbo.Klienci
WHERE
  nazwisko BETWEEN 'F' AND 'T'
```

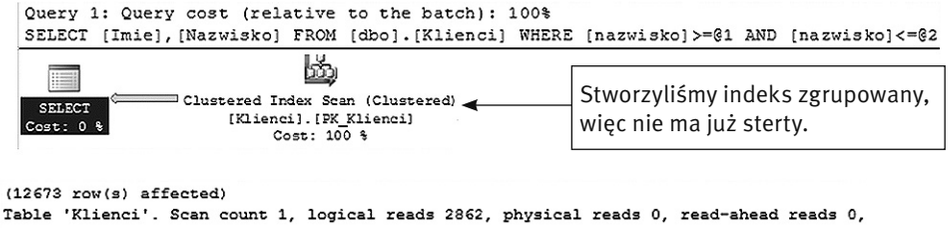
Po dwukrotnym wykonaniu zapytania uzyskujemy rezultaty:



Rysunek 16.
Plan wykonania – skanowanie tabeli

Zgodnie z oczekiwaniami, do realizacji zapytania wykorzystana została operacja skanowania tabeli. Przy pierwszym wykonaniu konieczne było pobranie stron danych z dysku (liczba fizycznych odczytów większa od 0). Każde następne wykonanie korzysta już ze stron umieszczonych w cache, czego przejawem jest zerowa wartość fizycznych odczytów. Całkowity koszt zapytania realizowanego według tego planu jest równy 2,1385.

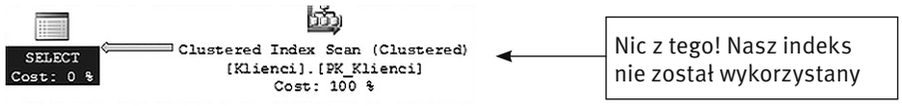
Pierwszym etapem naszych działań jest utworzenie indeksu zgrupowanego na kolumnie ID. Nie przyczyni się to w znaczącym stopniu do zwiększenia wydajności, ale spowoduje zmianę planu wykonania. Skoro utworzenie indeksu zgrupowanego powoduje fizyczne uporządkowanie stron danych (i likwidację sterty), to plan wykonania powinien zawierać wykonanie innej operacji niż skanowanie tabeli. Po wykonaniu zapytania stwierdzamy, że faktycznie tak jest, patrz rys. 17.



Rysunek 17.
Plan wykonania – skanowanie indeksu niezgrupowanego

Tym razem serwer skorzystał z operacji skanowania indeksu zgrupowanego. Nie jest to żaden skok wydajnościowy, bo tak czy siak przejrzeć trzeba wszystkie strony danych, gdyż nasze kryterium wyszukiwania nie jest kolumną zawartą w indeksie.

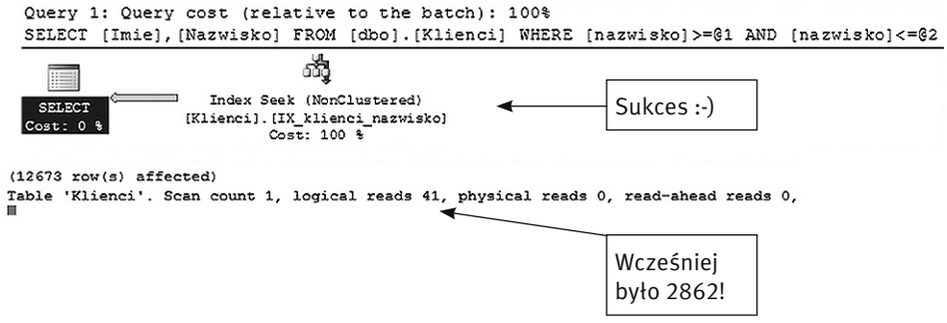
Rozpocznijmy teraz działania ukierunkowane na obniżenie kosztów realizacji zapytania. Pierwszy pomysł – stwórzmy indeks niezgrupowany na kolumnie nazwisko, która jest wykorzystywana jako kryterium wyszukiwania. Powinno to spowodować wykorzystanie tego indeksu do wyszukania wierszy z nazwiskami z określonego przez nas zakresu. Niestety po wykonaniu zapytania doznaliśmy rozczarowania:



Rysunek 18.
Plan wykonania – nie wykorzystane wyszukiwanie według indeksu

Plan wykonania się nie zmienił! Dlaczego? Z powodu umieszczenia na liście kolumn wyjściowych kolumny Imie. Optymalizator zapytań stwierdził, że mimo istnienia indeksu niezgrupowanego na kolumnie, po której wyszukujemy, nie warto z niego korzystać, gdyż i tak trzeba sięgnąć do stron danych, żeby pobrać wartości kolumny Imie. Z tego powodu plan wykonania nie uległ zmianie.

Skoro przemyśleliśmy już mechanizm działania zapytania i rolę indeksu, doprowadźmy sprawę do końca. Usuńmy istniejący indeks niezgrupowany, utwórzmy go na nowo z dodaną kolumną Imie. Po wykonaniu zapytania po raz kolejny, okazuje się, że tym razem indeks został wykorzystany:



Rysunek 19.
Plan wykonania – operacja wyszukiwania według indeksu

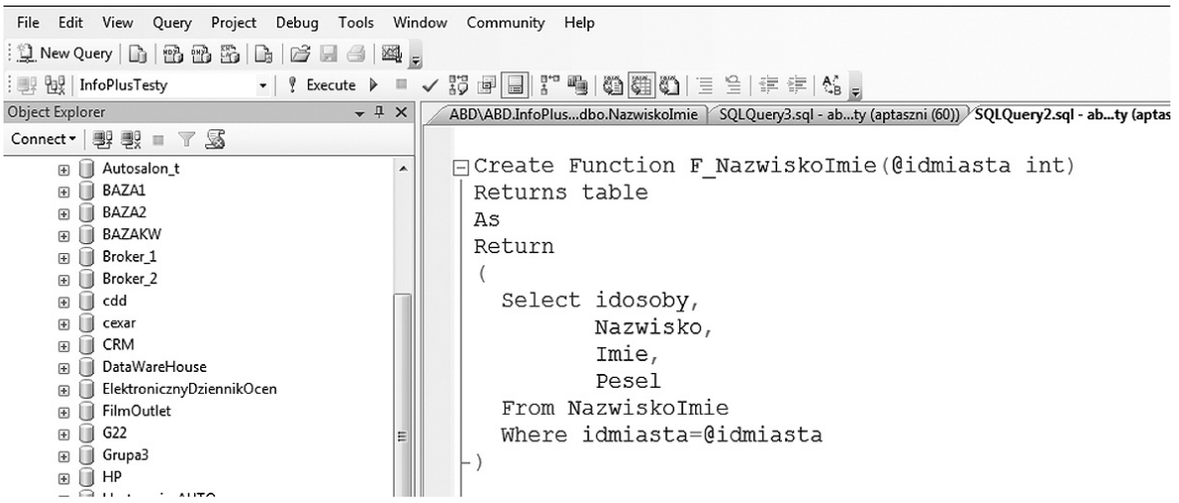
Odpowiednie wiersze spełniające kryteria wyszukiwania zostały zlokalizowane bardzo łatwo dzięki indeksowi niezgrupowanemu. Dodatkowo nie było konieczności sięgania do stron danych, gdyż indeks zawierał także kolumnę Imie, która była potrzebna do realizacji zapytania. Efekt jest widoczny. Koszt realizacji zapytania spadł z 2,1385 do 0,0453!

Po zakończeniu „walki" z optymalizacją prostego zapytania przez utworzenie odpowiednich indeksów można zdać sobie sprawę, iż wykonywanie tego typu operacji na prawdziwych bazach danych jest procesem złożonym i żmudnym. Do tego często nie da się pogodzić ze sobą wydajności dwóch lub więcej zapytań, bo każda poprawa wydajności w jednym psuje wydajność drugiego. Dodatkowo każdy kolejny indeks to dodatkowy problem z jego utrzymaniem oraz więcej czynności do wykonania przy modyfikacji danych. Jak sobie z tym radzić? Nie ma jednej sprawdzonej i zawsze działającej recepty. Są pewne podejścia pozwalające na realizację czynności w określonym porządku, co może się przyczynić do uniknięcia błędów lub ułatwienia wychwycenia typowych problemów. Zawsze jednak optymalizowanie wydajności pozostanie po części sztuką.

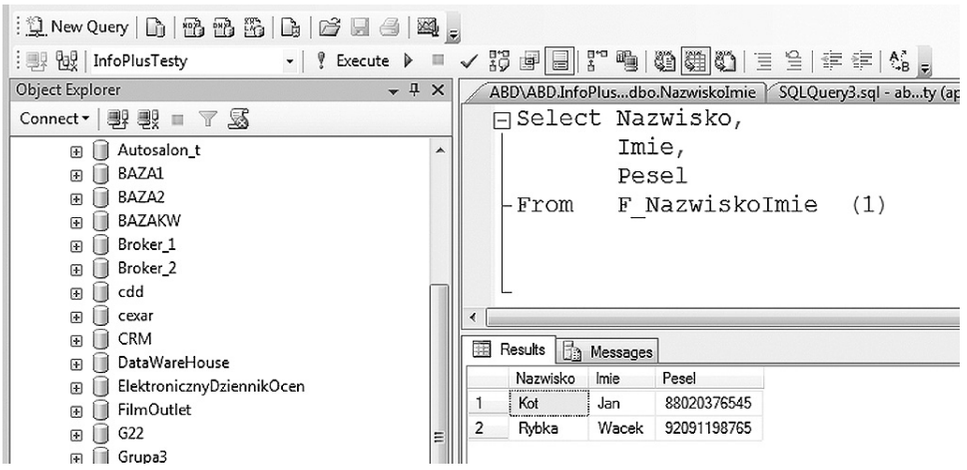
Ćwiczenie 7. **Programowanie funkcji tabelarycznej**

Ćwiczenie realizujemy według następującego schematu:

1. Aby wykonać zadanie, klikamy na przycisku New Query (nowe zapytanie) – uruchamia się okienko edycyjne, w którym będziemy wpisywać kod definiujący funkcję tabelaryczną (patrz rys. 20).
2. Wykonując zapisane polecenie klikamy na przycisku (poleceniu) Execute (lub naciskamy klawisz F5), patrz rys. 20.
3. Po poprawnym wykonaniu polecenia zdefiniowana funkcja jest gotowa do wykorzystania.
4. Sprawdzamy działanie funkcji wykonując zapytanie pokazane na rys. 21.
5. W kodzie wyzwalacza pokazanym na rys.20 i rys. 21, każdy uczestnik modyfikuje te miejsca kodu gdzie występuje słowo „NazwiskoImie", zastępując je swoim imieniem i nazwiskiem.
6. Sprawdzamy działanie funkcji zmieniając wartość parametru.



Rysunek 20.
Definiowanie funkcji tabelarycznej.



Rysunek 21.
Zapytanie wykorzystujące funkcje tabelaryczną wraz z przykładowym wynikiem

Ćwiczenie 8. **Pokaz tworzenia kopii bezpieczeństwa**

W ramach tego ćwiczenia, prowadzący zademonstruje tworzenie kopii zapasowych (BACKUP) oraz proces odtwarzania bazy danych z kopii zapasowej (RESTORE).

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