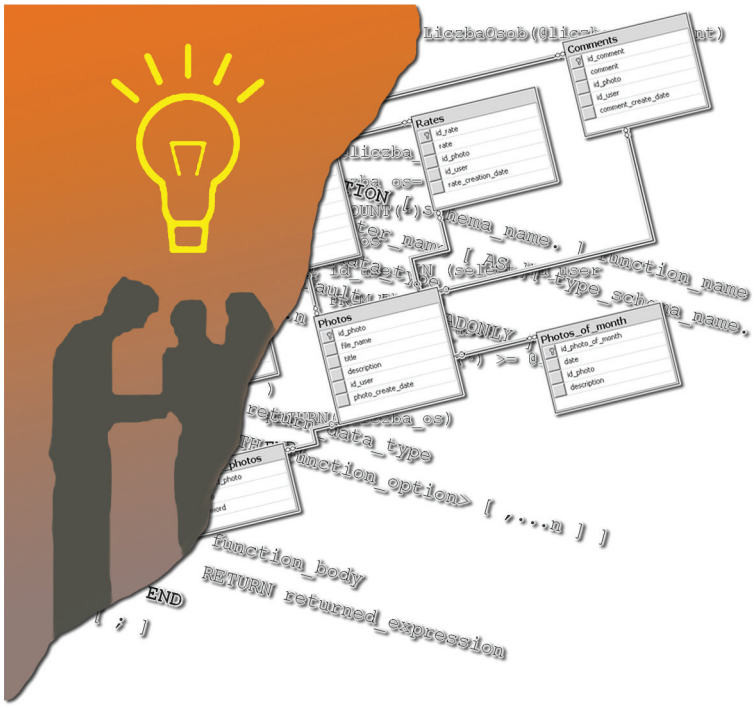
Maria Skublewska-Paszkowska
Małgorzata Plechawska-Wójcik



Współczesne Technologie Informatyczne

Wprowadzenie do programowania w języku SQL i T-SQL

w środowisku Microsoft SQL Server

Partnerzy:

WSPÓŁCZESNE TECHNOLOGIE INFORMATYCZNE

WPROWADZENIE DO PROGRAMOWANIA W JĘZYKU SQL I T-SQL

W ŚRODOWISKU MICROSOFT SQL SERWER

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Projekt Absolwent na miarę czasu współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

Wydział Elektrotechniki i Informatyki

Politechnika Lubelska
Wydział Elektrotechniki i Informatyki
ul. Nadbystrzycka 38A
20-618 Lublin

WSPÓŁCZESNE TECHNOLOGIE INFORMATYCZNE

WPROWADZENIE DO PROGRAMOWANIA W JĘZYKU SQL I T-SQL

W ŚRODOWISKU MICROSOFT SQL SERWER

Maria Skublewska-Paszkowska

Małgorzata Plechawska-Wójcik

Politechnika Lubelska
Lublin 2014

Recenzenci:
dr hab. Stanisław Grzegórski, prof. Politechniki Lubelskiej
dr inż. Grzegorz Kozieł, Politechnika Lubelska

Publikacja finansowana z projektu „Absolwent na miarę czasu"

Projekt „Absolwent na miarę czasu" współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego. Nr umowy UDA-POKL.04.01.01-00-421/10-01

Ta publikacja odzwierciedla jedynie stanowiska jej autorów, a Komisja Europejska nie ponosi odpowiedzialności za informacje w niej zawarte

Publikacja dystrybuowana bezpłatnie

Publikacja wydana za zgodą Rektora Politechniki Lubelskiej

ISBN: 978-83-7947-020-4

Wydawca: Politechnika Lubelska
ul. Nadbystrzycka 38D, 20-618 Lublin
Realizacja: Biblioteka Politechniki Lubelskiej
Ośrodek ds. Wydawnictw i Biblioteki Cyfrowej
ul. Nadbystrzycka 36A, 20-618 Lublin
tel. (81) 538-46-59, email: wydawca@pollub.pl
www.biblioteka.pollub.pl
Druk: TOP Agencja Reklamowa Agnieszka Łuczak
www.agencjatop.pl

Elektroniczna wersja książki dostępna w Bibliotece Cyfrowej PL www.bc.pollub.pl
Nakład: 100 egz.

Spis treści

Wstęp

Monografia „*Wprowadzenie do programowania w języku SQL i T-SQL w środowisku Microsoft SQL Server*" zawiera podstawowe informacje na temat języka SQL oraz jego rozszerzenia – Transact SQL. Skierowana jest do szerokiego grona odbiorców, w tym przede wszystkim do studentów kierunku Informatyka i pokrewnych, zarówno I jak i II stopnia.

W monografii przedstawiono szereg przykładów, które zostały wykonane w oparciu o utworzoną bazę danych w środowisku Microsoft SQL Server 2005. Wyjaśnione zostały też schemat ERD oraz najważniejsze relacje pomiędzy tabelami.

Cała książka została podzielona na dwie części. Pierwsza zawiera najważniejsze elementy języka SQL. Przedstawione zostały typy danych oraz podstawowe operatory stosowane w składni SQL. Wszystkie przykłady poleceń SQL były omawiane od najprostszych poleceń, do coraz bardziej zaawansowanych. Opisano także najpopularniejsze zmienne i funkcje systemowe.

Druga część książki została poświęcona tematyce konstruowania zapytań w języku Transact SQL (T-SQL). W tym zakresie przedstawiono podstawy, takie jak: tworzenie prostych programów, zmiennych oraz ich stosowanie. Bardziej zaawansowany materiał dotyczy tworzenia własnych funkcji i procedur w celu późniejszego korzystania już z utworzonych bloków programowych. Istotnym zagadnieniem, poruszanym w książce, jest tworzenie oraz użycie kursorów, dzięki którym możliwe jest przetwarzanie kolejnych rekordów zwróconych przez zapytanie.

Ostatni rozdział przedstawia wyniki analizy przeprowadzonej przez Autorki związanej z wydajnością stosowania języka T-SQL. Analiza dotyczyła zajętości procesora oraz czasu wykonywania: zapytań i wybranych poleceń języka T-SQL

Monografia powstała dzięki wysiłkowi autorów oraz recenzentów. Wszystkim tym, którzy przyczynili się do powstania tej książki, serdecznie dziękujemy.

Autorki
Małgorzata Plechawska-Wójcik
Maria Skublewska-Paszkowska

1

Struktura bazy danych

Cel

Rozdział przedstawia strukturę relacyjnej bazy danych, o którą opierać się będą opisy i przykłady zawarte w kolejnych rozdziałach książki. Ponadto rozdział ten zawiera omówienie poszczególnych tabel, ich atrybutów a także relacji zachodzących pomiędzy nimi.

1.1. STRUKTURA BAZY DANYCH

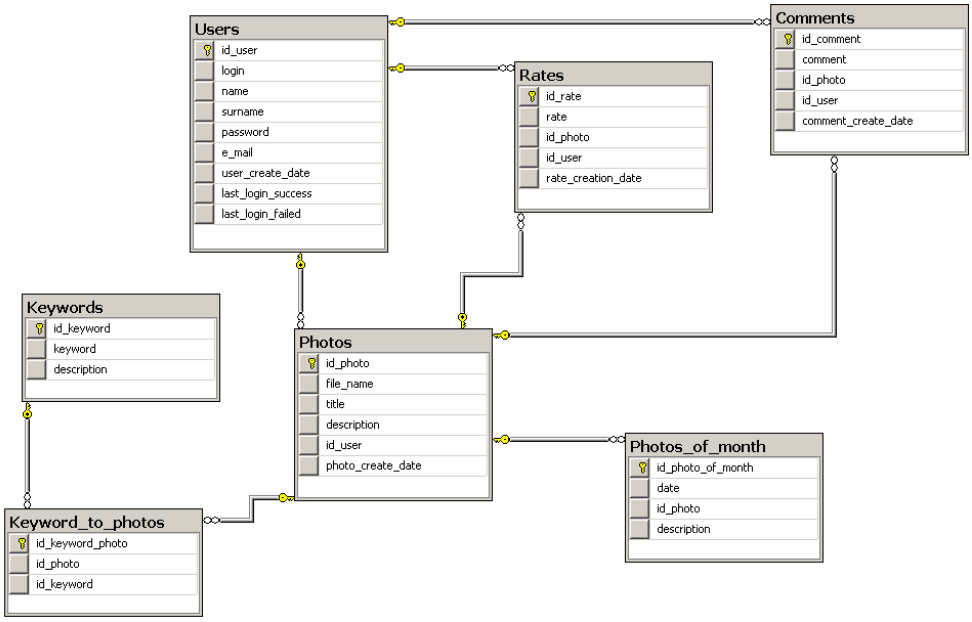
Diagram związków encji – ERD (ang. *Entity Relationship Diagram*) bazy danych został przedstawiony na rysunku 1.1. Przedstawia on encje, które reprezentują obiekty świata rzeczywistego. Każda encja musi mieć zdefiniowaną unikalną nazwę oraz zbiór atrybutów ją charakteryzujących. Dowolny obiekt może być reprezentowany poprzez tylko jedną encję (Łojewski, 2011).

Przykładem encji jest np. klient lub użytkownik. Atrybuty encji reprezentują własności poszczególnych wystąpień encji, przykładowo poszczególnych klientów czy użytkowników. Atrybuty encji obejmują nazwę, typ danych, opcjonalną unikalność oraz określoność (przyjmowanie wartości `NULL` lub `NOT NULL`). Występują dwa rodzaje atrybutów encji: identyfikatory oraz deskryptory (Łojewski, 2011). Identyfikator to unikalny atrybut lub zbiór atrybutów o wartościach `NOT NULL`, którego zadaniem jest jednoznaczna identyfikacja poszczególnych wystąpień encji. Deskryptor to atrybut niebędący identyfikatorem.

Diagram encji przedstawia również relacje, czyli związki łączące encje. Związki takie posiadają cechy ułatwiające zrozumienie modelu. Są to: stopień związku (liczba połączonych encji), liczność (liczba wystąpień poszczególnych powiązanych encji) oraz istnienie (opcjonalność lub obowiązkowość związku).

Przedstawiony diagram reprezentuje relacyjną bazę danych, gdzie poszczególne encje odwzorowane są w tabele, a atrybuty encji w kolumny. Wybrane unikalne identyfikatory encji są z kolei przekształcane w klucze główne.

Baza ta to zbiór połączonych ze sobą tabel zawierających dane użytkowników aplikacji przechowującej zdjęcia. Zdjęcia mogą być komentowane oraz oceniane. Przechowywane są także słowa kluczowe opisujące poszczególne zdjęcia, a także dane zdjęć wybranych jako zdjęcie miesiąca.



Rys. 1.1. Schemat ERD
Źródło: opracowanie własne

Tabele *Users*, *Photos*, *Photos_of_month*, *Rates*, *Comments*, *Keywords* zawierają dane odpowiednio: użytkowników, zdjęć, zdjęć miesiąca, ocen, komentarzy i słów kluczowych. Każda tabela przechowuje zestaw wartości zdefiniowanych przez poszczególne kolumny. Każdy kolejny wiersz tabeli zawiera dane konkretnego obiektu, wystąpienia encji.

Każda z tabel posiada dokładnie jedną kolumnę reprezentującą klucz główny tabeli. Przykładowo dla tabeli *Users* kluczem głównym jest `id_user`, dla tabeli *Photos* – `id_photo`. Wartości kluczy głównych muszą być unikalne i określone (przechowujące wartości `NOT NULL`), co gwarantuje jednoznaczną identyfikowalność poszczególnych wierszy.

Poza kluczami głównymi w poszczególnych tabelach znajdują się kolumny przechowujące własności poszczególnych wierszy. Przykładowo tabela *Keywords*, poza kluczem głównym (kolumna `id_keyword`) przechowuje także właściwe słowo kluczowe (kolumna `keyword`) oraz krótki opis (`description`). Analogicznie tabela *Photos* przechowuje następujące dane zdjęć: identyfikator (`id_photo`), nazwa pliku (`file_name`), tytuł (`title`), opis (`description`), datę utworzenia (`photo_create_date`). Tabela *Photos* zawiera dodatkowo kolumnę `id_user`. Kolumna ta przechowuje wartości klucza obcego i umożliwia tym samym określenie relacji pomiędzy tabelami *Photos* i *Users*.

Wartości będące kluczami obcymi muszą odpowiadać wartościom kluczy głównych przechowywanych w złączonej tabeli. Taki sposób przechowywania umożliwia realizację relacji pomiędzy tabelami. Wartości kluczy obcych jednej tabeli odpowiadają wartościom kluczy głównych w drugiej. Przykładem takiej relacji jest związek pomiędzy tabelami *Users* oraz *Photos*. Tabela *Photos* przechowuje

w kolumnie `id_user` klucze obce, których wartości odpowiadają wartościom kluczy głównych tabeli *Users*. Dane z obu tabel przedstawiają odpowiednio tabela 1.1. (dane użytkowników z tabeli *Users*) oraz tabela 1.2. (dane zdjęć z tabeli *Photos*).

Tabela 1.1. Wybrane kolumny tabeli Users

id_user	***login***	***name***	***surname***	***user_create_date***	***last_login_success***
1	mkowalski	Maciej	Kowalski	2011-03-10 12:10:00.000	2012-03-10 12:10:00.000
2	anowicka	Anna	Nowicka	2011-08-10 13:11:00.000	2012-03-10 13:15:00.000
3	nborowiec	Natalia	Borowiec	2012-12-11 08:05:00.000	2012-01-11 08:08:00.000
4	pnowak	Patryk	Nowak	2012-03-15 07:45:00.000	2012-03-15 07:55:00.000
5	inowak	Iwona	Nowak	2012-02-20 10:11:00.000	2012-02-20 10:11:00.000
6	mwojcik	Magdalena	Wójcik	2012-03-20 00:00:00.000	2012-03-21 00:00:00.000
7	mkowalczyk	Marek	Kowalczyk	2012-03-29 10:13:00.000	2012-03-29 10:23:00.000
8	mkucharczyk	Maria	Kucharczyk	2012-04-01 11:05:00.000	NULL
9	amikulska	Anna	Mikulska	2012-04-01 13:15:00.000	NULL
10	kdebski	Karol	Debski	2012-04-05 17:45:00.000	2012-04-05 17:52:00.000
11	akot	Artur	Kot	2012-04-12 09:21:00.000	2012-04-12 09:24:00.000
12	jkimak	Joanna	Kimak	2012-07-14 08:12:00.000	2012-07-14 08:12:15.000
13	amucha	Anna	Mucha	2012-09-12 18:12:00.000	2012-09-12 20:12:00.000
14	bjakubiak	Beata	Jakubiak	2012-10-17 10:30:00.000	NULL
15	akepa	Agnieszka	Kepa	2012-10-17 10:45:00.000	NULL
16	mpiatek	Michal	Piatek	2012-10-18 18:13:00.000	2012-10-18 18:21:00.000

Tabela 1.2. Wybrane kolumny z tabeli Photos

id_photo	***file_name***	***title***	***id_user***	***Photo_create_date***
1	zachodslonca1.png	Zachód slonca nad Wisla	1	2011-12-10 11:23:00.000
2	split.jpg	Pocztówka z Chorwacji	2	2012-03-12 08:31:00.000
3	ogrod.jpg	Mój ogród	3	2012-03-30 10:05:00.000
4	modelarstwo.png	Konferencja pasjonatów modelarstwa	4	2012-03-21 11:05:00.000
5	bonzai.jpg	Wystawa roslin Bonzai	3	2012-02-22 20:15:00.000
6	pekin.png	Wspomnienia z podróży	6	2012-01-15 13:17:00.000
7	tybet.png	Wspomnienia z podróży cd	6	2012-01-15 13:25:00.000
8	mongolia.png	Wspomnienia z podróży cd	6	2012-01-15 13:27:00.000
9	mur_chinski.png	Wspomnienia z podróży cd	6	2012-01-15 13:20:00.000
10	amelka.jpg	Nasza córeczka	7	2012-03-15 12:21:00.000
11	pluto.png	Nasz jamnik	8	2012-04-01 09:11:00.000
12	w_ciemnosci.jpg	W ciemnosci – nowy firm A.Holland	1	2012-01-11 11:05:00.000
13	hm.png	Wiosenna kolekcja firmy H&M	13	2012-04-02 14:12:00.000
14	torebki.jpg	Nowa kolekcja firmy Kazar	13	2012-03-11 13:12:00.000
15	asus.png	Najnowszy tablet firmy Asus	12	2012-02-13 15:09:00.000
16	brokuly.jpg	Zapiekanka z brokulami	14	2012-04-15 12:21:00.000
17	fikus.png	Nowy kwiatek	15	2012-03-11 13:12:00.000
18	kaktusy.jpg	Kaktusy duze i male	16	2012-03-28 16:12:00.000
19	grecja.png	Nasze plany wakacyjne	15	2012-01-15 13:25:00.000
20	warsztaty.png	Warsztaty taneczne w Lublinie	15	2012-01-22 08:15:00.000

Relacja pomiędzy tabelami *Users* i *Photos* jest relacją jeden-do-wielu (1:N). Oznacza to, iż jeden użytkownik może dodać wiele zdjęć. Jednak jedno zdjęcie może być dodane jedynie przez konkretnego użytkownika. Ze względu na taką relację, klucz obcy umieszczony został w tabeli *Photos*. Pozwala na łatwe zdefiniowanie identyfikatora użytkownika, który dodał określone zdjęcie. Klucz obcy dla tej relacji nie mógłby być umieszczony w tabeli *Users*, ponieważ jedna kolumna nie pozwoliłaby na zdefiniowanie wielu zdjęć użytkownika. Regułą jest, iż w przypadku relacji jeden-do-wielu klucze obce powinny być umieszczane w tabeli, która w relacji jest po stronie „wiele".

Warto zauważyć, iż klucze obce w tabeli *Photos* nie są unikalne. Jeśli określony użytkownik dodał wiele zdjęć (np. użytkownik o identyfikatorze 15 zamieścił 3 zdjęcia), jego identyfikator pojawi się w wielu wierszach. W przypadku użytkownika o identyfikatorze 15 – jego identyfikator pojawi się w kolumnie `id_user` dokładnie przy trzech zdjęciach: tych o identyfikatorach 17,19 i 20.

Podobne relacje zachodzą pomiędzy tabelami:

- *Users* i *Comments* (jeden użytkownik może dodać wiele komentarzy)
- *Users* i *Rates* (jeden użytkownik może dodać wiele ocen)
- *Photos* i *Comments* (jedno zdjęcie może mieć wiele komentarzy)
- *Photos* i *Rates* (jedno zdjęcie może mieć wiele ocen)
- *Photos* i *Photos_of_month* (jedno zdjęcie może zostać zdjęciem miesiąca wiele razy)

Tabela 1.3. Tabela Keywords

id_keyword	***keyword***	***descripton***
1	podróże	Dalekie podróże i wypady za miasto
2	konferencje	Spotkania mniej lub bardziej formalne
3	moda	Ubrania, buty, torebki itp
4	sprzet	
5	komputery	komputery, laptopy i urzadzenia mobilne
6	RTV i AGD	NULL
7	rosliny	NULL
8	zwierzeta	dzikie i domowe
9	rodzina	NULL
10	kulinaria	przepisy, nowe pomysly
11	natura	piekne krajobrazy
12	miasta	NULL
13	kino	Nowosci w kinie, kreacje aktorskie

Zdefiniowanie relacji pomiędzy tabelami *Photos* oraz *Keywords* wymagało dodania tabeli przechodniej, określonej na diagramie jako *Keywords_to_Photos*. Zależność pomiędzy zdjęciami oraz słowami kluczowymi jest zdefiniowana następująco: zdjęcie może być opisane wieloma słowami kluczowymi, pojedyncze słowo kluczowe może opisywać wiele zdjęć. Zależność taka jest relacją wiele-do-wielu (N:M), która w praktyce musi zostać zastąpiona dwiema relacjami typu jeden-do-wielu (1:N, M:1). Operację taką przeprowadzono poprzez umieszczenie pomiędzy tabelami

Photos i *Keywords* tabeli *Keywords_to_Photos*, która zawiera jedynie klucz główny oraz klucze obce reprezentujące zarówno zdjęcia (`id_photo`) jak i słowa kluczowe (`id_keyword`). Wprowadzenie tej tabeli umożliwiło zdefiniowanie dwóch relacji jeden-do-wielu i tym samym rozbicie relacji wiele-do-wielu.
Tabele 1.3. i 1.4. zawierają dane odpowiednio z tabel *Keywords* oraz *Keywords_to_Photos*. Wszystkie zdefiniowane w przedstawionej bazie danych relacje mają stopień binarny, ponieważ każda z nich występuje pomiędzy dwiema tabelami.

Tabela 1.4. Tabela Keywords_to_photos

id_keyword_photo	***id_photo***	***Id_keyword***
1	1	1
2	1	11
3	2	1
4	2	12
5	3	7
6	4	2
7	5	2
8	5	7
9	6	1
10	6	12
11	7	1
12	8	11
13	8	1
14	9	1
15	10	9
16	11	8
17	12	12
18	13	3
19	14	3
20	15	4
21	15	5
22	16	10
23	17	7
24	18	7
25	18	2
26	19	1
27	20	2

1.2. PYTANIA KONTROLNE

1. Wymień i opisz cechy relacji.
2. Wyjaśnij, jak w praktyce postępuje się z relacjami typu N:M.
3. Wyjaśnij różnicę pomiędzy identyfikatorem a deskryptorem.

2

Typy danych i operatory

Cel

Celem rozdziału jest przedstawienie typów danych dostępnych w środowisku MS SQL Server. Podzielone są one na siedem grup. Pośród omówionych typów znajdują się typy liczbowe, typy daty i czasu a także znakowe oraz dedykowane obiektom bazodanowym.

Plan

1. Typy danych
2. Operatory

2.1. TYPY DANYCH

Typ danych jest definiowany jako atrybut określający rodzaj i zakres danych, jakie może przyjmować obiekt. W MS SQL Server typy danych związane są zarówno z kolumnami jaki i zmiennymi, wyrażeniami oraz parametrami (MSDN, Data Types). Język T-SQL daje też użytkownikowi możliwość definiowania własnych typów danych.

Typy danych w MS SQL Server można podzielić na siedem grup:

- Liczby dokładne (ang. *Exact Numbers*)
- Liczby aproksymowane (ang. *Approximate Numbers*)
- Typy daty i czasu (ang. *Date and Times*)
- Ciągi znaków (ang. *Character Strings*)
- Ciągi znaków Unicode (ang. *Unicode Character Strings*)
- Ciągi binarne (ang. *Binary Strings*)
- Inne typy danych

Typy liczbowe podzielić można na dwie grupy. Pierwsza z nich to liczby o dokładnych wartościach (tabela 2.1.), druga – liczby zmiennoprzecinkowe o jedynie przybliżonej dokładności (tabela 2.2.).

Tabela 2.1. Typy danych – liczby dokładne

nazwa	***zakres***	***rozmiar***	***opis***
bigint	<-2^63; 2^63-1>	8 bajtów	Typ całkowity
bit	[0;1]	1 bit	Typ zero-jedynkowy
decimal	<-10^38+1; 10^38-1>	w zależności od precyzji	Typ o ustalonej precyzji i skali
int	<-2^31; 2^31-1>	4 bajty	Typ całkowity
money	<-922.337.203.685.477,5808; 922.337.203.685.477,5807>	8 bajtów	Typ reprezentujący waluty
numeric	<-10^38+1; 10^38-1>	w zależności od precyzji	Typ o ustalonej precyzji i skali
smallint	<-2^15; 2^15-1>	2 bajty	Typ całkowity
smallmoney	<- 214.748,3648; 214.748,3647>	4 bajty	Typ reprezentujący waluty
tinyint	<0; 255>	1 bajt	Typ całkowity dodatni

Tabela 2.2. Typy danych – liczby aproksymowane

nazwa	***zakres***	***rozmiar***	***opis***
float [(n)]	<-1.79E+308; -2.23E-308>, 0,<2.23E-308; 1.79E+308>	4 lub 8 bajtów – w zależności od precyzji (7 lub 15)	Typ zmiennoprzecinkowy o liczbie bitów określanej w parametrze (1 – 53)
real	<-3.40E+38; -1.18E-38>, 0,<1.18E-38;3.40E+38>	4 bajty	Typ zmiennoprzecinkowy – float(24)

Typy daty i czasu są w MS SQL rozbudowane. Jest ich sześć rodzajów (tabela 2.3.). Ich rozumienie jest istotne dla poprawnego przechowywania, konwertowania oraz formatowania tego typu danych.

Tabela 2.3. Typy danych – data i czas

nazwa	***Przykładowy format***	***rozmiar***	***opis***
date	YYYY-MM-DD	3 bajty	Data z dokładnością do 1 dnia
datetime	YYYY-MM-DD hh:mm:ss:[nnn]	8 bajtów	Data i czas z dokładnością do ułamków sekund (max 3 cyfry), zegar 24-ro godzinny
datetime2	YYYY-MM-DD hh:mm:ss:[nnnnnnn]	6-8 bajtów (w zależności od precyzji)	Data i czas z dokładnością do ułamków sekund (max 7 cyfr), zegar 24-ro godzinny
datetimeoffset	YYYY-MM-DD hh:mm:ss[nnnnnnn] [{+|-}hh:mm]	10 bajtów	Data i czas z dokładnością do ułamków sekund (max 7 cyfr) z uwzględnieniem strefy czasowej, zegar 24-ro godzinny
smalldatetime	YYYY-MM-DD hh:mm:ss	4 bajty	Data i czas z dokładnością do 1 sekundy, zegar 24-ro godzinny
time	hh:mm:ss:[nnnnnnn]	5 bajtów	Czas z dokładnością do ułamków sekund (max 7 cyfr), zegar 24-ro godzinny

Kolejną grupą są ciągi znaków. Grupa ta zawiera trzy typy bardzo przydatne w codziennej pracy z bazą danych. Zestawienie typów z tej grupy przedstawia tabela 2.4. Typy te mają kodowanie inne niż Unicode.

Tabela 2.4. Typy danych – ciągi znaków

nazwa	***długość***	***opis***
char [(n)]	n – wartość pomiędzy 1 a 8.000	Ciąg znaków o określonej długości (n)
text	maksymalna długość – 2^31-1 bajtów	Ciąg znaków o zmiennej długości
varchar [(n | max)]	n – wartość pomiędzy 1 a 8.000 max – maksymalny rozmiar – 2^31-1 bajtów (2 GB)	Ciąg znaków o zmiennej długości (n)

Grupa ciągów znaków Unicode zawiera trzy typy danych będące odpowiednikami (dla kodowania Unicode) tych przedstawionych w tabeli 2.4. Szczegółowy opis zawiera tabela 2.5.

Tabela 2.5. Typy danych – ciągi znaków Unicode

nazwa	***długość***	***opis***
nchar [(n)]	n – wartość pomiędzy 1 a 4.000. Maksymalny rozmiar wynosi 2n	Ciąg znaków o określonej długości (n)
ntext	maksymalna długość – 2^30-1 bajtów	Ciąg znaków o zmiennej długości. Aktualnie użyty rozmiar (w bajtach) jest dwukrotnie większy od aktualnej długości ciągu.
nvarchar [(n | max)]	n – wartość pomiędzy 1 a 4.000 max – maksymalny rozmiar – 2^31-1 bajtów (2 GB)	Ciąg znaków o zmiennej długości (n). Aktualnie użyty rozmiar (w bajtach) jest dwukrotnie większy od aktualnej długości ciągu + 2 bajty.

Grupa zawierająca typy ciągów binarnych również zawiera trzy typy danych. Przedstawione one zostały w tabeli 2.6.

Tabela 2.6. Typy danych – binarne ciągi znaków

nazwa	***długość***	***opis***
binary [(n)]	n – wartość pomiędzy 1 a 8.000	Dane binarne o określonej długości (n)
image	maksymalna długość – 2^31-1 bajtów	Dane binarne o zmiennej długości
varbinary [(n | max)]	n – wartość pomiędzy 1 a 8.000 max – maksymalny rozmiar – 2^31-1 bajtów (2 GB)	Dane binarne o zmiennej długości (n)

Tabela 2.7 przedstawia pozostałe typy danych dostępnych w środowisku MS SQL Server. Typy te służą do przechowywania obiektów bazodanowych lub specyficznych formatów danych i są wykorzystywane w języku Transact-SQL.

Tabela 2.7. Typy danych – pozostałe typy

nazwa	***opis***
cursor	Typ przechowujący referencję do obiektu kursor.
hierarchyid	Typ danych o zmiennej długości reprezentujący pozycję w hierarchii.
sql_variant	Specjalny typ danych przechowujący dane różnych typów wspieranych przez SQL Server (takie jak int, binary czy char). Maksymalna długość: 8016 bajtów. Typ ten nie przechowuje jednak danych następujących typów: datetimeoffset, geography, geometry, hierarchyid, image, ntext, nvarchar, rowversion, sql_variant, text, varbinary, varchar, xml.
table	Typ danych wykorzystywany do tymczasowego przechowywania zbioru wierszy pobranych z tabeli.
rowversion	Typ danych wspierający auto-generowanie unikalnych, auto-inkrementowanych numerów wykorzystywanych w oznaczaniu kolejnych wierszy tabeli.
uniqeidentifier	Unikalny, 16-sto bitowy typ przechowujący wartość GUID.
xml	Typ danych dedykowany przechowywaniu danych XML.

2.2. OPERATORY

Operator jest symbolem, który precyzuje określoną operację wykonywaną na jednym lub wielu wyrażeniach. Operatory można pogrupować w następujące kategorie (MSDN, Operators):

- Arytmetyczne,
- Logiczne,
- Porównania,
- Przypisania,
- Bitowe,
- Konkatenacji,
- Jednoargumentowe.

Arytmetyczne operatory umożliwiają wykonywanie podstawowych operacji matematycznych. Podstawowe operatory zostały przedstawione w tabeli 2.8.

Tabela 2.8. Operatory arytmetyczne (MSDN, Operators)

operator	***znaczenie***
+	Dodawanie
-	Odejmowanie
/	Dzielenie
*	Mnożenie
%	Operator modulo

Operatory dodawania oraz odejmowania mogą zostać użyte na zmiennych także typu daty (datetime, smalldatetime).

Operatory logiczne zostały przedstawione w tabeli 2.9. Operacje wykonywane z użyciem tych operatorów zwracają wartość typu Boolean, jedną z trzech możliwych: TRUE, FALSE oraz UNKNOWN.

Tabela 2.9. Operatory logiczne (MSDN, Operators)

operator	***znaczenie***
ALL	Zwraca wartość TRUE, jeśli porównanie każdego elementu ze zbioru jest prawdziwe
AND	Zwraca wartość TRUE, jeśli wszystkie porównywane wartości logiczne mają wartość prawda
ANY	Zwraca wartość TRUE, jeśli chociaż jeden element zbioru pasuje do porównywanej wartości
BETWEEN	Zwraca wartość TRUE, jeśli element porównywany zawiera się w rozpatrywanym przedziale
EXISTS	Zwraca wartość TRUE, jeśli podzapytanie zwraca jakikolwiek wiersz
IN	Zwraca wartość TRUE, jeśli porównywalna wartość jest równa z jednym elementem listy
LIKE	Zwraca wartość TRUE, jeśli porównywalna wartość pasuje do wzorca
NOT	Zaprzecza logicznej wartości
OR	Zwraca wartość TRUE, jeśli przynajmniej jedna wartość logiczna jest prawdziwa
SOME	Zwraca wartość TRUE, jeśli niektóre elementy zbioru mają wartość TRUE

Kolejną grupą operatorów są operatory porównania. Dzięki nim sprawdzane są relacje zachodzące pomiędzy wyrażeniami. Operatory mogą zostać zastosowane na wszystkich wyrażeniach, z wyjątkiem tych opartych o zmienne typu tekstowego (text, ntext) oraz obrazowych. Podstawowe operatory porównania zostały przedstawione w tabeli 2.10.

Tabela 2.10. Operatory porównania (MSDN, Operators)

operator	***znaczenie***
=	równy
>	większy niż
<	mniejszy niż
>=	większy równy
<=	mniejszy równy
<>	różny
!=	różny (nie dla standardu SQL-92)
!>	nie większy niż (nie dla standardu SQL-92)
!<	nie mniejszy niż (nie dla standardu SQL-92)

Ważnym elementem jest operator przypisania. Dzięki niemu można nadać zmiennej wartość zgodną z jej typem. Może on być także użyty w instrukcji SQL. Operator ten jest reprezentowany przez znak równości (=).

Operatory bitowe stosowane są do wykonywania bitowych operacji na wyrażeniach będącymi liczbami całkowitymi. Podstawowe operatory zostały zebrane w tabeli 2.11.

Tabela 2.11. Operatory bitowe (MSDN, Operators)

operator	***znaczenie***
&	operator bitowy AND
I	operator bitowy OR
^	operator bitowy XOR

Związek elementów porównywanych (prawy i lewy) został przedstawiony w tabeli 2.12.

Tabela 2.12. Zależności pomiędzy prawymi i lewymi operandami (MSDN, Operators)

Lewy operand	***Prawy operand***
binary	int, smallint, tinyint
bit	int, smallint, tinyint, bit
int	int**,** smallint**,** tinyint**,** binary**,** varbinary
smallint	int, smallint, tinyint, binary, varbinary
tinyint	int, smallint, tinyint, binary, varbinary
varbinary	int, smallint, tinyint

Operator konkatenacji, czyli łączenia, stosowany jest na danych typu tekstowego. Operator konkatenacji to znak plus (+). Konkatenację można także wykonać korzystając z funkcji CONCAT (MSDN, Operators).

Ostatnią grupę stanowią operatory jednoargumentowe. Operatory te są używane na wyrażeniach i zmiennych typu numerycznego. Jedynie operator negacji bitowej można zastosować wyłącznie na danych typu całkowitego. Operatory jednoargumentowe zostały zebrane w tabeli 2.13.

Tabela 2.13. Operatory jednoargumentowe i ich znaczenie (MSDN, Operators)

Operator	***Znaczenie***
-	Pozytywna wartość
+	Ujemna wartość
~	Negacjabitowa

2.3. PYTANIA KONTROLNE

1. Wyjaśnij podział numerycznych typów danych.
2. Wymień i omów różnice pomiędzy poszczególnymi typami daty i czasu.
3. Wyjaśnij zastosowanie typu danych *rowversion*.

3

Proste zapytania

Cel

Rozdział zawiera podstawowe informacje związane z prostym wyszukiwaniem informacji z bazy danych poprzez stosowanie zapytania `SELECT`, a także zarządzanie otrzymanymi rekordami, przy użyciu podstawowych klauzul. Proste wyszukiwanie polega na pobieraniu wszystkich lub wybranych danych z jednej tabeli. Istnieje możliwość zmiany nazw wyświetlanych kolumn przy pomocy aliasów. Polecenie `DISTINCT` zapewnia eliminacje powtarzających się rekordów pobranych z bazy. Rekordy mogą zostać także posortowane według zadanych kryteriów rosnąco lub malejąco.

Plan

1. Struktura zapytań wybierających
2. Aliasy
3. Polecenie `DISTINCT`
4. Sortowanie

3.1. WSTĘP

Jedną z podstawowych operacji wykonywanych na bazie danych jest wydobywanie z niej informacji. Wszystkie lub wybrane dane można odczytać przy pomocy zapytań języka SQL (ang. *Structured Query Language*). Za jego pomocą można nie tylko pobrać wszystkie dane z tabeli bazy danych, ale także wpływać na kolejność wyświetlania pobieranych rekordów, jak również wpływać na nazwy zwracanych kolumn za pomocą aliasów. Wybrane przez zapytanie rekordy można sortować według zadanych kryteriów.

Język SQL używany jest nie tylko przez osoby uczące się czy początkujących programistów, ale także osoby zawodowo zajmujące się tworzeniem i eksploatacją systemów bazodanowych.

3.2. STRUKTURA ZAPYTAŃ WYBIERAJĄCYCH

Podstawowy schemat polecenia pobierającego dane z bazy, czyli instrukcji `SELECT`, zostało przedstawione na listingu 3.1. Zapytanie składa się ze słowa kluczowego `SELECT` (MSDN, Writing SQL Queries). Po nim następuje wyliczenie nazw kolumn, których dane zostaną pobrane. W przypadku, gdy mają zostać wyświetlone wszystkie dane z wybranej tabeli, zamiast wymienić kolejne kolumny, wystarczy użyć znaku *. W takim przypadku kolumny zostaną wyświetlone według kolejności ich tworzenia. Jeśli użytkownik chce wyświetlić dane w określonym porządku, powinien wyliczyć nazwy kolumn. Po poleceniu `SELECT` można także umieszczać wyrażenia. Po słowie kluczowym `FROM` należy wymienić nazwy tabel, z których będą pobierane dane. Po klauzuli `WHERE` podawane są ograniczenia wykonywanego zapytania (np. wyszukiwanie osób, których nazwiska rozpoczynają się literą *P*), a także złączenia tabel, jeśli dane są pobierane z większej liczby tabel niż jedna. Po słowie kluczowym `ORDER BY` można podać kolumny, według których zostanie zastosowane sortowanie. Zapytanie może być zakończone średnikiem.

Listing 3.1. Podstawowa struktura zapytania `SELECT`

```
SELECT kolumna 1 [, kolumna 2] …[, kolumna n] | *
FROM tabela 1 [, tabela 2]… [, tabela n]
WHERE ograniczenie
ORDER BY sortowanie;
```

Listing 3.2. Struktura zapytania `SELECT` *(MSDN ,SELECT Clause)*

```
SELECT [ ALL | DISTINCT ]
[ TOP ( expression ) [ PERCENT ] [ WITH TIES ] ]
<select_list>
<select_list> ::=
    {
      *
      | { table_name | view_name | table_alias }.*
      | {
          [ { table_name | view_name | table_alias }. ]
               { column_name | $IDENTITY | $ROWGUID }
          | udt_column_name [ { . | :: } { { property_name
          | field_name }
            | method_name ( argument [ ,...n] ) } ]
          | expression
          [ [ AS ] column_alias ]
         }
      | column_alias = expression
    } [ ,...n ]
```

Na listingu 3.2. została przedstawiona szczegółowa struktura zapytania `SELECT`. Podstawowe jej składowe to (MSDN ,SELECT Clause):

- `ALL` – klauzula wyświetlająca wszystkie rekordy (domyślna wartość),
- `DISTINCT` – klauzula eliminująca powtarzające się wiersze wyników,
- `TOP (expression ) [ PERCENT ] [ WITH TIES ]` – klauzula określająca jaki procent wierszy wynikowych powinny zostać wyświetlone,
- `< select_list >` – lista kolumn, z których zostaną wyświetlone dane (maksymalna wartość wyrażeń podana po klauzuli `SELECT` to 4096),
- * – symbol wyświetlający wszystkie kolumny z tabeli (tabel) podanej po klauzuli `FROM`,
- `table_name | view_name | table_alias.*` – zakres danych związanych z symbolem *,
- `column_name` – nazwa kolumny do zwrócenia,
- `$IDENTITY` –numer identyfikacyjny kolumny (w przypadku większej liczby należy podać kwalifikacyjną nazwę, czyli poprzedzoną nazwą kolumny),
- $`ROWGUID` – zwraca wiersz `GUID` kolumny.

Wyświetlenie wszystkich danych z tabeli *Keywords* zostało pokazane na listingu 3.3. Ponieważ użyto znaku * pobrane zostają wszystkie dane z tej tabeli.

Listing 3.3. Pobranie danych z tabeli Keywords

```
SELECT *
FROM Keywords;
```

Źródło: opracowanie własne

Zapytanie przedstawione na listingu 3.3. zwraca wszystkie rekordy z tabeli *Keywords*. Tabela ta zawiera 12 rekordów, które zostały zebrane w tabeli 3.1. Jako nagłówki widoczne są nazwy poszczególnych kolumn tej tabeli. W każdym wierszu znajduje się jeden rekord zwróconych danych, w tym przypadku rekord tabeli.

Tabela 3.1. Wynik zapytania z listingu 3.3.

Id_keyword	***keyword***	***Description***
1	podróże	Dalekie podróże i wypady za miasto
2	konferencje	Spotkania mniej lub bardziej formalne
3	moda	Ubrania, buty, torebki itp.
4	sprzęt	
5	komputery	komputery, laptopy i urządzenia mobilne
6	RTV i AGD	NULL
7	rośliny	NULL
8	zwierzęta	dzikie i domowe
9	rodzina	NULL
10	kulinaria	przepisy, nowe pomysły
11	natura	piękne krajobrazy
12	miasta	NULL

Źródło: opracowanie własne

Pierwszą wyświetlaną kolumną jest `id_keyword`. Ma ona przypisany atrybut klucza głównego (`PRIMARY KEY`) tabeli *Keywords*. Druga kolumna zawiera słowa kluczowe, a ostatnia ich opis. Tam, gdzie podczas dodawania danych pominięto daną wartość, została wyświetlona wartość `NULL`.

W przypadku, gdy należy wyświetlić dane pochodzące z jednej kolumny tabeli, wystarczy po słowie kluczowym `SELECT` napisać nazwę kolumny, co przedstawiono na listingu 3.4. Zwrócony wynik (tabela 3.2.) zawiera jedynie nazwy wszystkich słów kluczowych, które zostały wprowadzone do bazy danych.

Listing 3.4. Pobranie danych z kolumny keyword tabeli Keywords

```
SELECT keyword
FROM Keywords;
```

Źródło: opracowanie własne

Tabela 3.2. Wynik zapytania z Listingu 3.4.

keyword
podróże
konferencje
moda
sprzęt
komputery
RTV i AGD
rośliny
zwierzęta
rodzina
kulinaria
natura
miasta

Źródło: opracowanie własne

Wyświetlenie z tabeli *Keywords* danych z więcej niż jednej kolumny tabeli możliwe jest poprzez wypisanie kolejnych nazw kolumn oddzielonych przecinkami. Przykład wyświetlający słowo kluczowe i odpowiadający mu opis został przedstawiony na listingu 3.5. Dane zwracane w wyniku zapytania ustawione są według podanej kolejności.

Listing 3.5. Pobranie danych z tabeli Keywords

```
SELECT keyword, description
FROM Keywords;
```

Źródło: opracowanie własne

3.3. ALIASY

Domyślnie, w wyniku wykonania zapytania wybierającego z wyszczególnionymi kolumnami, zwrócone zostają kolumny, których nazwy są zatytułowane tak samo, jak istniejące nazwy kolumn w tabeli. Stosując alias, można jednak podać inną nazwę,

która zostanie wyświetlona jako nazwa kolumny zwróconych danych. Alias w zapytaniu podaje się w dwojaki sposób: po słowie `AS` (MSDN ,SELECT Clause) lub bezpośrednio po spacji, po nazwie wyświetlanej kolumny. Jeśli aliasem ma być ciąg zawierający spację, należy ująć go w cudzysłów. Przykład zastosowania aliasów został przedstawiony na listingu 3.6. Zapytanie to zwróci wynik przedstawiony w tabeli 3.3.

Listing 3.6. Wyświetlenie danych z aliasami kolumn

```
SELECT keyword AS "Słowo klucz", description AS "Opis"
FROM Keywords;
```

Źródło: opracowanie własne

Tabela 3.3. Wynik zapytania z Listingu 3.6.

Słowo klucz	***Opis***
podróże	Dalekie podróże i wypady za miasto
konferencje	Spotkania mniej lub bardziej formalne
moda	Ubrania, buty, torebki itp
sprzęt	
komputery	komputery, laptopy i urządzenia mobilne
RTV i AGD	NULL
rośliny	NULL
zwierzęta	dzikie i domowe
rodzina	NULL
kulinaria	przepisy, nowe pomysły
natura	piękne krajobrazy
miasta	NULL

Źródło: opracowanie własne

3.4. POLECENIE DISTINCT

Istnieje możliwość eliminacji wyświetlania wierszy powtarzających się. Przykładem może być wyświetlenie nazwisk z tabeli *Users*. Istnieje prawdopodobieństwo, że przechowywane w tabeli nazwiska osób mogą się powtórzyć. Oznacza to, że w bazie danych mogą być przechowywane dane osób o takich samych nazwiskach. W takim przypadku, podczas wykonywania prostego zapytania omawianego dotychczas, zostaną wyświetlone wszystkie nazwiska, także te, które występują wielokrotnie. Natomiast zastosowanie klauzuli `DISTINCT`, eliminującej wyświetlanie powtarzających się wierszy, pozwala na otrzymanie unikalnych wyników. Przykład użycia tej klauzuli został przedstawiony na listingu 3.7.

Listing 3.7. Zapytanie z klauzulą `DISTINCT`

```
SELECT DISTINCT surname AS Nazwisko
FROM Users;
```

Źródło: opracowanie własne

W tabeli 3.4. zostało przedstawione porównanie wyniku zwróconego przez zapytanie z listingu 3.7., jak również wynik zapytania bez klauzuli eliminującej powtarzające się wyniki.

Tabela 3.4. Porównanie zapytań z oraz bez klauzuli `DISTINCT`

Rekordy zapytania z klauzulą DISTINCT	***Rekordy zapytania bez klauzuli DISTINCT***
Nazwisko	***Nazwisko***
Borowiec	Kowalski
Dębski	Nowicka
Jakubiak	Borowiec
Kępa	Nowak
Kimak	Nowak
Kot	Wójcik
Kowalczyk	Kowalczyk
Kowalski	Kucharczyk
Kucharczyk	Mikulska
Mikulska	Dębski
Mucha	Kot
Nowak	Kimak
Nowicka	Mucha
Piątek	Jakubiak
Wójcik	Kępa
	Piątek
	Kowalski

Źródło: opracowanie własne

Pierwsze zapytanie z klauzulą `DISTINCT` zwraca 15 rekordów, podczas gdy zapytanie bez omawianej klauzuli wyświetla o 2 rekordy więcej. Oznacza to, że występują powtarzające się nazwiska, a są nimi: *Kowalski* oraz *Nowak*.

3.5. SORTOWANIE

Dane, które są pobierane z poszczególnych tabel czy widoków, mogą zostać posortowane według zadanych kryteriów. Struktura klauzuli sortowania (polecenie `ORDER BY`) została przedstawiona na listingu 3.8.

Listing 3.8. Schemat klauzuli sortującej (MSDN, ORDER BY)

```
ORDER BY
    {
        order_by_expression [SKIP n] [LIMIT n]
        [ COLLATE collation_name ]
        [ ASC | DESC ]
    }
    [ ,…n ]
```

Źródło: opracowanie własne

Kolejne elementy klauzuli sortującej to (MSDN, ORDER BY):

- `order_by_expression` – zapytanie precyzujące dane do sortowania,
- `COLLATE {collation_name}` – precyzuje, czy klauzula `ORDER BY`, ma być wykonywana zgodnie z parametrem `collation_name`. `COLLATE` jest dedykowana dla wyrażeń tekstowych,
- `ASC` – precyzuje porządek sortowania rosnąco (opcja domyślna),
- `DESC` – precyzuje porządek sortowania malejąco,
- `LIMIT n` – określa liczbę wierszy do wyświetlenia,
- `SKIP` *n* – określa liczbę wierszy do pominięcia.

Wyświetlenie posortowanych danych wszystkich użytkowników prezentuje listing 3.9. W pierwszej kolejności dane zostały posortowane alfabetycznie według nazwiska, a w drugiej według imienia w odwrotnej kolejności. Wyświetlone rekordy zostały przedstawione w tabeli 3.5.

Listing 3.9. Sortowanie wyświetlanych danych

```
SELECT surname Nazwisko, name Imię
From Users
ORDER BY surname, name desc;
```

Źródło: opracowanie własne

Tabela 3.5. Posortowane dane z tabeli Users

Nazwisko	***Imię***
Borowiec	Natalia
Dębski	Karol
Jakubiak	Beata
Kępa	Agnieszka
Kimak	Joanna
Kot	Artur
Kowalczyk	Marek
Kowalski	Stanislaw
Kowalski	Maciej
Kucharczyk	Maria
Mikulska	Anna
Mucha	Anna
Nowak	Patryk
Nowak	Iwona
Nowicka	Anna
Piątek	Michał

Źródło: opracowanie własne

Działanie klauzuli sortującej jest głównie widoczne przy nazwiskach: *Kowalski* oraz *Nowak*.

3.6. PYTANIA KONTROLNE

1. Omów strukturę zapytania wybierającego.
2. Do czego służy operator * w poleceniach wybierających?
3. Do czego służą aliasy?
4. Co powoduje klauza `DISTINCT` w zapytaniach wybierających?
5. W jaki sposób można sortować rekordy?

4

Zapytania z klauzulą WHERE

Cel

Rozdział przedstawia tworzenie zapytań w języku SQL z zastosowaniem klauzuli `WHERE`. Zaprezentowano w nim użycie tej klauzuli, zarówno do ograniczania wyszukiwanych danych, jak i do pobierania danych z wielu tabel. Ograniczenie wyszukiwania tabel polega na podawaniu różnych warunków, które muszą zostać spełnione, aby rekordy zostały wyświetlone. Dodatkowo można łączyć wiele tabel przy pomocy relacji zachodzących między tabelami na podstawie klucza głównego oraz kluczy głównych występujących w innych tabelach. Przedstawione opisy, przykłady i ich rezultaty omawiają zastosowanie klauzuli `WHERE` w zapytaniach wybierających.

Plan

1. Ograniczanie wyszukiwanych danych
2. Łączenie danych

4.1. WSTĘP

Jedną z podstawowych klauzul w zapytaniu wybierającym jest opcja `WHERE`, która pełni dwojaką funkcję. Po pierwsze służy jako klauzula, po której podawane są dodatkowe ograniczenia względem danych wyszukiwanych i podanych po poleceniu `SELECT`. Jej druga funkcjonalność umożliwia pobieranie danych z większej liczby tabel niż jedna na podstawie ich złączenia, które podawane jest właśnie w tej klauzuli. Klauzula ta jest bardzo często używana, ponieważ zapewnia większą elastyczność wyszukiwanych rekordów. Jest ona stosowana w praktyce, nie tylko w prostych zapytaniach.

4.2. OGRANICZANIE WYSZUKIWANYCH DANYCH

Ograniczenie wyników realizowane jest poprzez podanie różnego rodzaju warunków w klauzuli `WHERE` (MSDN, WHERE). Warunki te mogą składać się z operacji warunkowych, równości, czy też innych operatorów takich jak: `IN`, `NOT IN`, `LIKE` itp.

Zapytanie wyświetlające dane osób, które noszą nazwisko *Kowalski* przedstawione jest na listingu 4.1. Po słowie kluczowym `WHERE` należy podać nazwę kolumny tabeli, która zawiera wyszukiwany ciąg znaków (w tym przykładzie nazwisko – *surname*). Należy pamiętać, że wyszukiwany jest dokładny ciąg znaków, z uwzględnieniem wielkości liter.

Listing 4.1. Zapytanie wyświetlające osoby o nazwisku Kowalski

```
SELECT surname, name, login, e_mail, user_create_date
FROM Users
WHERE surname = 'Kowalski';
```

Źródło: opracowanie własne

Klauzula `WHERE` z listingu 4.1 ogranicza wszystkie dane użytkowników znajdujące się w tabeli *Users* do tych, których nazwiska są identyczne z ciągiem podanym w apostrofach. Rekordy, które zostały zwrócone przez powyższe zapytanie (ograniczone do kolumn: nazwisko, imię, login, email oraz datę utworzenia użytkownika), zostały przedstawione w tabeli 4.1.

Tabela 4.1. Dane osób o nazwisku Kowalski

surname	***name***	***login***	***e_mail***	***user_create_date***
Kowalski	Maciej	mkowalski	mkowalski@email.pl	2011-03-10 12:10:00.000
Kowalski	Stanisław	skowalski	skowal@o2.pl	2012-04-22 20:36:54.867

Źródło: opracowanie własne

Zapytanie z listingu 4.1 przedstawia użycie operatora równości w celu ograniczenia wyników do podanego ciągu. Istnieje także możliwość wyświetlenia danych, które rozpoczynają się od zadanego ciągu lub znaku. Na listingu 4.2 został przedstawiony przykład wyświetlający dane osób, których nazwiska rozpoczynają się od litery *N*. Do tego celu został użyty operator `LIKE`, po którym należy podać ciąg znaków do wyszukania. W przypadku zapytania z listingu 4.2, szukany jest dowolny ciąg znaków rozpoczynający się literą *N*, czyli '*N%*', gdzie % oznacza wystąpienie dowolnego ciągu znaków po literze *N*.

Listing 4.2. Zapytanie wyświetlające osoby o nazwisku zaczynające się literą N

```
SELECT surname, name, login, e_mail, user_create_date
FROM Users
WHERE surname Like 'N%';
```

Źródło: opracowanie własne

W tabeli 4.2 został przedstawiony wynik, który uzyskano po wykonaniu zapytania z listingu 4.2.

Tabela 4.2. Dane osób o nazwisku rozpoczynającym się literą N

surname	***name***	***login***	***e_mail***	***user_create_date***
Nowicka	Anna	anowicka	anowicka@e.pl	2011-08-10 13:11:00.000
Nowak	Patryk	pnowak	pnowak@gm.com	2012-03-15 07:45:00.000
Nowak	Iwona	inowak	inowak@nn.eu	2012-02-20 10:11:00.000

Źródło: opracowanie własne

4.3. ŁĄCZENIE TABEL

Bardzo często zachodzi potrzeba wyświetlenia danych z więcej niż jednej tabeli. W przypadku, gdy dane są ze sobą powiązane, do ich złączenia w zapytaniu można zastosować klauzulę `WHERE` (MSDN, WHERE). Po niej należy podać istniejące powiązania pomiędzy tabelami. Powiązanie pomiędzy tabelami jest realizowane na

podstawie więzów integralności dotyczących relacji klucza głównego z kluczem obcym. Wartość klucza głównego jednej tabeli występuje jako wartość w innej tabeli jako klucz obcy. Istnieje związek pomiędzy liczbą tabel z których dane są odczytywane, a relacjami między nimi (Ullman 2011). Minimalna liczba relacji jest o jeden mniejsza niż liczba tabel, co przedstawia schemat na listingu 4.3. Przykład ten ilustruje zapytanie wybierające dane z *p* kolumn z różnych tabel (od *tabela_1* do *tabela_n*). Istniejące relacje są określone po klauzuli `WHERE`. W przypadku, gdy kolumny z kilku tabel mają identyczną nazwę, należy podać pełną nazwę z użyciem separatora, czyli nazwy kolumny poprzedzoną nazwą tabeli w której się znajduje (*nazwa_tabeli.nazwa_kolumny*).

Listing 4.3. Zapytanie z klauzulą `WHERE` służącą do łączenia tabel

```
SELECT  kol_1, kol_2, …, kol_p
FROM tabela_1, tabela_2, ... , tabela_n
WHERE relacja_1 AND relacja_2  AND ... AND relacja_n-1
```

Źródło: opracowanie własne

Dane o fotografiach, które zostały zakwalifikowane jako zdjęcia miesiąca można uzyskać wykonując zapytanie przedstawione na listingu 4.4. Wyświetlone zostały tam takie dane jak: identyfikator zdjęcia miesiąca, identyfikator zdjęcia, tytuł zdjęcia, jego opis, a także identyfikator użytkownika, który je dodał. Przy podawaniu nazw kolumn należy pamiętać, że muszą one zostać określone jednoznacznie. Oznacza to, że nazwa kolumny musi być unikalna. Ponieważ nazwy kolumn mogą się powtarzać w obrębie różnych tabel, należy podać nazwę bezwzględną (np. p.id_photo).

Po klauzuli `FROM` należy podać wszystkie nazwy tabel, z których dane zostaną pobrane. Na listingu 4.4 dodatkowo zastosowano aliasy, dzięki czemu skraca się bezwzględne nazwy kolumn, jak i warunki relacji. Ponieważ główne dane o zdjęciach miesiąca są zapisane w tabeli *Photos*, a dodatkowe w tabeli *Photos_of_month*, należało podać te dwie nazwy.

Po klauzuli `WHERE` można zdefiniować relacje. Ponieważ dane pochodzą z dwóch tabel, wystarczy podać jedną relację, która zawiera powiązanie klucz główny tabeli *Photos* z kluczem obcym tabeli *Photos_of_month*. Należy wyraźnie wskazać, która kolumna należy do której tabeli. Przydatne tu okazały się aliasy podane po klauzuli `FROM`.

Listing 4.4. Zapytanie wyświetlające dane zdjęcia miesiąca

```
SELECT id_photo_of_month, p.id_photo, title, p.description,
      id_user
FROM Photos p, Photos_of_month pm
WHERE p.id_photo = pm.id_photo;
```

Źródło: opracowanie własne

Rezultat wykonania zapytania z listingu 4.4 został przedstawiony w tabeli 4.3. Zwróconych zostało 5 rekordów.

Tabela 4.3. Wyniki zapytania z listingu 4.4.

id_photo	***id_photo_of_month***	***title***	***description***	***id_user***
1	1	Zachód słońca nad Wisłą	Wspomnienie z wakacji	1
8	2	Wspomnienia z podróży cd	Piękne krajobrazy	6
5	3	Wystawa roślin Bonzai	2-dniowa wystawa, ponad 1000 roślin. 10-11 kwietnia 2012	3
18	4	Kaktusy duże i małe	Zapraszamy na wystawę kaktusów	16

Źródło: opracowanie własne

Gdyby w przykładzie z listingu 4.4 wynikła potrzeba wyświetlenia dodatkowo danych użytkownika, który dodał zdjęcie miesiąca, należałoby dodać do zapytania jeszcze jedną tabelę *Users*. Wtedy, po klauzuli `FROM` należy dopisać kolejną tabelę, z której zostaną pobrane dane, a także dodać jeszcze jeden warunek relacji zachodzącej pomiędzy tabelą nowododaną a pozostałą (`p.id_user = u.id_user`). Oczywiście obie relacje muszą być spełnione, więc są one powiązane spójnikiem logicznym `AND`. Przykład zmodyfikowanego zapytania wybierającego, rozbudowanego o wyświetlenie loginu użytkownika przedstawiono na listingu 4.5. Zapytanie skonstruowano na podstawie kodu przedstawionego na listingu 4.4. Pobrane rekordy zostały przedstawione w tabeli 4.4.

Listing 4.5. Zapytanie wyświetlające dane zdjęcia miesiąca rozszerzone o login użytkownika

```
SELECT id_photo_of_month, p.id_photo, title, p.description,
       login
FROM Photos p, Photos_of_month pm, Users u
WHERE p.id_photo = pm.id_photo and p.id_user = u.id_user;
```

Źródło: opracowanie własne

Porównując wyniki przedstawione w tabeli 4.3 oraz w tabeli 4.4 można stwierdzić, że są to te same dane, z tą różnicą, że ostatnia kolumna została zamieniona z identyfikatora użytkownika na jego login.

Tabela 4.4. Wyniki zapytania z listingu 4.5.

id_photo_of_month	id_photo	title	description	login
1	1	Zachód słońca nad Wisłą	Wspomnienie z wakacji	mkowalski
2	8	Wspomnienia z podróży cd	Piękne krajobrazy	mwojcik
3	5	Wystawa roślin Bonzai	2-dniowa wystawa, ponad 1000 roślin. 10-11 kwietnia 2012	nborowiec
4	18	Kaktusy duże i małe	Zapraszamy na wystawę kaktusów	mpiatek

Źródło: opracowanie własne

Dane odczytane z wielu tabel, podobnie jak z jednej, można sortować według zadanego kryterium. W tym celu należy dodać klauzulę `ORDER BY`, po której umieszcza się dane według których wybrane rekordy zostają posortowane. Przykład takiego zapytania został przedstawiony na listingu 4.6. Zapytanie to wyświetla dane zdjęć miesiąca: identyfikator, tytuł, opis, imię i nazwisko użytkownika. Dane te zostały posortowane według daty dodania zdjęć rosnąco (jest to domyślny rodzaj sortowania). Wynik zapytania z listingu 6. został przedstawiony w tabeli 4.5.

Listing 4.6. Zapytanie wyświetlające dane zdjęć miesiąca pogrupowane według daty dodania

```
SELECT id_photo_of_month, title, p.description, name,
       surname, date
FROM Photos p, Photos_of_month pm, Users u
WHERE p.id_photo = pm.id_photo and p.id_user = u.id_user
ORDER BY date;
```

Źródło: opracowanie własne

Tabela 4.5. Wyniki zapytania z listingu 4.6.

id_photo_of_month	***title***	***description***	***name***	***surname***	***date***
1	Zachód słońca nad Wisłą	Wspomnienie z wakacji	Maciej	Kowalski	2011-12-01
2	Wspomnienia z podróży cd	Piękne krajobrazy	Magdalena	Wójcik	2012-01-01
3	Wystawa roślin Bonzai	2-dniowa wystawa, ponad 1000 roślin. 10-11 kwietnia 2012	Natalia	Borowiec	2012-02-01
4	Kaktusy duże i małe	Zapraszamy na wystawę kaktusów	Michał	Piątek	2012-03-01

Źródło: opracowanie własne

W klauzuli `WHERE` mogą znajdować się jednocześnie warunki ograniczające zapytanie jak i relacje łączące tabele. Listing 4.7 przedstawia zapytanie wybierające wszystkie identyfikatory słów kluczowych oraz ich opisy dla zdjęcia, którego identyfikator jest równy 6. Dodatkowe ograniczenie, jakie zostało nałożone to takie, że istnieje opis słowa kluczowego. Ponieważ kolumna *description* tabeli *Keywords* dopuszcza pominięcie wstawienia wartości w rekordzie (wartość `NULL`), istnieje możliwość wyświetlenia danych, które zawierałyby słowo *`NULL`* zamiast konkretnej wartości (w tym przypadku tekstu).

Listing 4.7. Zapytanie wyświetlające identyfikatory i opisy słów kluczowych dodanych dla zdjęcia o identyfikatorze 6

```
SELECT kp.id_keyword_photo, description
FROM Keyword_to_photos kp, Keywords k
WHERE kp.id_keyword = k.id_keyword and id_photo = 6
      and description is not null;
```

Źródło: opracowanie własne

W tabeli 4.6. zostały umieszczone wyniki zapytania bez dodatkowego ograniczenia, natomiast w tabeli 4.7 został pokazany rekord zwrócony w wyniku wykonania pełnego zapytania z listingu 4.7. Można zauważyć, że jeśli przy dodawaniu słowa kluczowego nie został podany opis, w wyniku wykonania zapytania, zostanie zwrócony rekord zawierający wartość `NULL`. W celu wyeliminowania rekordów z taką wartością został dodany warunek przedstawiony na listingu 4.7. (`description is not null`) oraz połączony z pozostałymi warunkami operatorem logicznym `AND`.

Tabela 4.6. Wyniki zapytania z listingu 4.7. z pominięciem ograniczenia wyświetlenia rekordów z wartością `NULL`

id_keyword_photo	***description***
9	Dalekie podróże i wypady za miasto
10	NULL

Źródło: opracowanie własne

Tabela 4.7. Wyniki zapytania z listingu 4.7.

id_keyword_photo	***description***
9	Dalekie podróże i wypady za miasto

Źródło: opracowanie własne

Kolejny przykład zapytania korzystającego z klauzuli `WHERE` został przedstawiony na listingu 4.8. Zapytanie to wyświetla rekordy zawierające ocenę zdjęć, ich tytuły oraz loginy właścicieli zdjęć. Zostało dodane dodatkowe ograniczenie na wyświetlenie tylko tych zdjęć, które zostały ocenione na minimum 3 punkty, a maksymalnie na 6. Dodatkowo wszystkie rekordy zostały posortowane malejąco ze względu na uzyskaną ocenę (*rate*).

Listing 4.8. Zapytanie wyświetlające posortowane zdjęcia, ich oceny oraz użytkowników

```
SELECT rate, title, login
FROM Rates r, Photos p, Users u
WHERE r.id_photo = p.id_photo and r.id_user = u.id_user
      and rate >=3 and rate <=6
ORDER BY rate desc;
```

Źródło: opracowanie własne

Tabela 4.8. Wyniki zapytania z listingu 4.8.

rate	***title***	***login***
5	Zachód słońca nad Wisłą	mwojcik
5	Zachód słońca nad Wisłą	mkowalczyk
5	Mój ogród	mwojcik
5	Wspomnienia z podróży	mkowalski
5	Wspomnienia z podróży cd	mkowalski
5	Wspomnienia z podróży cd	anowicka
5	Wspomnienia z podróży cd	anowicka
5	Wspomnienia z podróży cd	inowak
5	Wspomnienia z podróży cd	mkowalski
5	Kaktusy duże i małe	jkimak
5	Kaktusy duże i małe	bjakubiak
5	W ciemności – nowy firm A.Holland	anowicka
4	Warsztaty taneczne w Lublinie	amikulska
4	W ciemności – nowy firm A.Holland	amikulska
4	Wspomnienia z podróży	anowicka
4	Wspomnienia z podróży cd	mkowalski
4	Pocztówka z Chorwacji	mkowalski
3	Mój ogród	amikulska
3	Wiosenna kolekcja firmy H&M	kdebski
3	Zachód słońca nad Wisłą	pnowak
3	Zachód słońca nad Wisłą	mkowalski
3	Nowa kolekcja firmy Kazar	mkowalczyk
3	Nowa kolekcja firmy Kazar	mkucharczyk

Źródło: opracowanie własne

Rekordy zwrócone w wyniku wykonania zapytania listingu 4.8, zostały przedstawione w tabeli 4.8. Zapytanie zwróciło 23 rekordy.

4.4. PYTANIA KONTROLNE

1. Do czego służy klauzula `WHERE` w zapytaniach SQL?
2. W jaki sposób można wyszukać dane dopasowane do podanego wzorca?
3. Na czym polega pobieranie danych z wielu tabel?
4. Jakie warunki muszą zostać spełnione przy pobieraniu danych z wielu tabel?
5. Jakiej klauzuli należy użyć w celu posortowania otrzymanych rekordów?

5

Złączenia

Cel

Łączenie danych umieszczonych w relacyjnej bazie danych to jedno z podstawowych zadań konstruowania zapytań. Dane, które podczas projektowania bazy danych zostały umieszczone w oddzielnych tabelach muszą zostać odpowiednio połączone, aby możliwe było swobodne z nich korzystanie. Rozdział przedstawia konstrukcje, sposoby użycia i przykłady różnych typów złączeń.

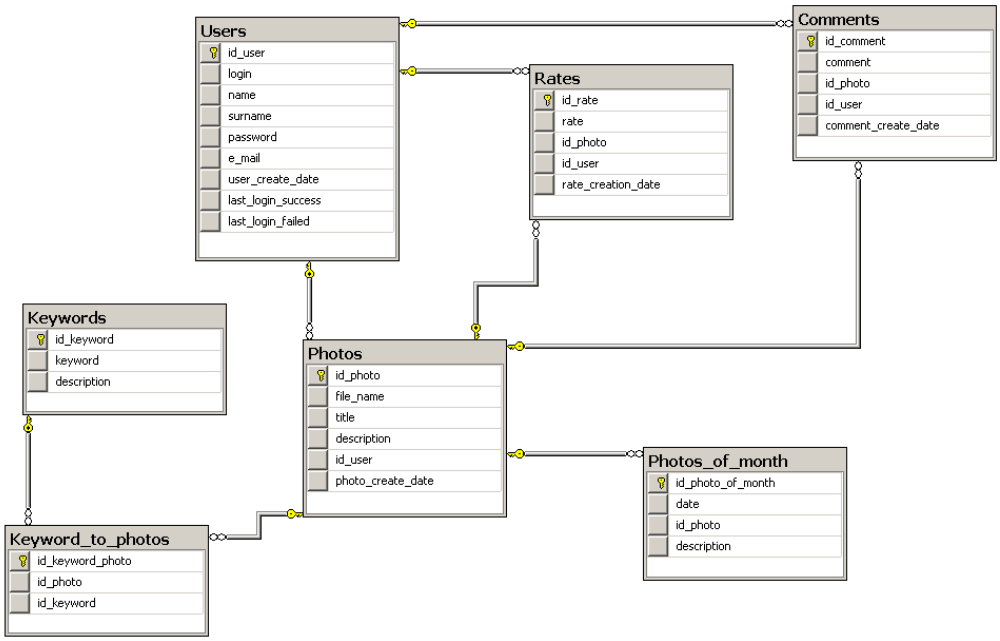
Plan

1. Złączenia w oparciu o warunki `WHERE`
2. Złączenia po klauzuli `FROM` (`JOIN`)
3. Dodatkowe operacje oparte o klauzulę `JOIN`
4. Złączenia typu `UNION`

5.1. WSTĘP

Istnieje kilka sposobów na utworzenie złączeń tabel w zapytaniach. Są to konstrukcje JOIN, złączenia oparte o klazulę FROM i WHERE, nierównozłączenia czy operatory takie jak UNION (Turley, Wood, 2009).

Na schemacie bazy danych (rysunek 5.1.) widać złączenia pomiędzy tabelami w postaci linii łączącej odpowiednie pola poszczególnych tabel. Pola te mają dodatkowe atrybuty – klucz główny (będący identyfikatorem wierszy w tabeli) lub klucz obcy (służący do definiowania relacji z inną tabelą). Klucze są to zwykle pojedyncze pola (jak w przykładzie z rysunku 5.1.), jednak można również zdefiniować klucze oparte na kilku polach.



Rys. 5.1. Schemat ERD
Źródło: opracowanie własne

Przykładem powiązania tabel może być relacja pomiędzy tabelami *Users* i *Photo*. W tabelach każdy wiersz posiada unikalny identyfikator (klucz główny) – dla tabeli *Users* przechowywany jest on w kolumnie `id_user`, dla tabeli *Photo* – w kolumnie `id_photo`. Tabela *Users* jest powiązana z tabelą *Photo*. Relację tę można zdefiniować następująco: jeden użytkownik może dodać wiele zdjęć, jednak określone zdjęcie przyporządkowane jest do jednego użytkownika (który je dodał).

Aby oznaczyć zależność, w tabeli *Photo* utworzona jest dodatkowa kolumna – `id_user` przechowująca identyfikatory użytkowników. Kolumna `id_user` jest więc kluczem obcym służącym do wskazania właściciela dla każdego zdjęcia. Kolumna ta umożliwia realizację relacji. Przykład ten opisuje typową w relacyjnych bazach danych relację jeden-do-wielu (1:n). Ponadto warto zauważyć, że klucze obce zawsze muszą mieć swój odpowiednik w zbiorze kluczy głównych.

W relacyjnych bazach danych spotyka się także relacje jeden-do-jednego (1:1). Relacje wiele-do-wielu (n:m) w praktyce nie występują – należy zastąpić je dwiema relacjami jeden-do-wielu dodając przy tym dodatkową tabelę, tzw. tabelę łączącą (DeBetta i in, 2008). Przykładem takiej relacji jest zależność pomiędzy tabelami *Keywords* i *Photos* – jedno zdjęcie może być oznaczone kilkoma słowami kluczowymi. Podobnie pojedyncze słowo kluczowe może (a nawet powinno) być przypisane do wielu zdjęć. Rozwiązaniem jest wprowadzenie tabeli łączącej *Keyword_to_photos*, która zawiera tabele `id_keyword` i `id_photo`, będące kluczami obcymi definiującymi relację *Keyword_to_photos* z tabelami *Keywords* i *Photos*.

Można wyróżnić trzy podstawowe sposoby na złączenie dwóch różnych tabel. Dwa z nich to różne opcje wykonania złączenia w pojedynczym zapytaniu (z wykorzystaniem klauzuli `WHERE` lub `FROM`). Trzecim sposobem jest zastosowanie podzapytania.

5.2. ZŁĄCZENIA W OPARCIU O WARUNKI WHERE

Ten tradycyjny sposób tworzenia złączeń jest w wielu środowiskach uznawany za przestarzały. Mimo tego jest on nadal wspierany, choć od wersji SQL Server 2008 nie jest już rekomendowany.

Ten typ złączenia konstruuje się następująco (DeBetta i in, 2008):

1. Po klauzuli `FROM` należy umieścić nazwy tabel, które mają być połączone.
2. Jeśli według schematu ERD tabele umieszczone w pt. 1 nie są ze sobą bezpośrednio połączone, do listy wyrażeń po klauzuli `FROM` należy dołączyć tabele pośrednie, przez które możliwe będzie utworzenie złączenia.
3. Po klauzuli `WHERE` należy umieścić warunki złączenia poszczególnych tabel. Pojedynczy warunek może dotyczyć złączenia dwóch tabel. Warunek zwykle jest przyrównaniem klucza głównego z jednej tabeli z odpowiadającym mu kluczem obcym z drugiej.
4. Jeżeli nazwy kolumn w warunku nie są jednoznaczne, należy je poprzedzić nazwą tabeli. Nazwę tabeli i nazwę kolumny należy rozdzielić przy pomocy kropki (.).
5. Kolejne warunki po klauzuli `WHERE` należy łączyć operatorem logicznym `AND`.

Przykład zapytania ze złączeniem dwóch tabel przedstawia listing 5.1. Fragment wyniku umieszczony został w tabeli 5.1. (oryginalnie zapytanie zwróciło 20 wierszy).

Listing 5.1. Przykład konstrukcji złączenia dwóch tabel

```
SELECT Users.id_user, login, id_photo, title
FROM Users, Photos
WHERE Users.id_user=Photos.id_user
```

Źródło: opracowanie własne

Tabela 5.1. Wynik zapytania z listingu 5.1.

	id_user	*login*	*id_photo*	*title*
1	1	mkowalski	1	Zachód słońca nad Wisłą
2	2	anowicka	2	Pocztówka z Chorwacji
3	3	nborowiec	3	Mój ogród
4	4	pnowak	4	Konferencja pasjonatów modelarstwa
5	3	nborowiec	5	Wystawa roślin Bonzai
6	6	mwojcik	6	Wspomnienia z podróży

Źródło: opracowanie własne

Znak równości w warunku złączenia oznacza, że zapytanie zwróci tylko te wiersze, w których występuje dokładna zgodność rekordów z obu tabel. Pominięte są więc np. rekordy z jednej tabeli, które nie mają swoich odpowiedników w drugiej lub ich klucze obce przyjmują wartość `NULL`.

Bardziej złożony przykład prezentuje listing 5.2. W zapytaniu są uwzględnione trzy tabele: *Photos*, *Keywords* i *Keyword_to_photos*, ponieważ zgodnie z diagramem ERD tabele *Photos* i *Keywords* nie są bezpośrednio połączone i do ich złączenia w zapytaniu należy wykorzystać również tabelę pośrednią *Keyword_to_photos*. Wynik przedstawia tabela 5.2.

Listing 5.2. Przykład konstrukcji złączenia trzech tabel

```
SELECT Photos.id_photo, title, keyword
FROM Photos, Keywords, Keyword_to_photos
WHERE Photos.id_photo=Keyword_to_photos.id_photo
AND Keywords.id_keyword=Keyword_to_photos.id_keyword
```

Źródło: opracowanie własne

Tabela 5.2. Wynik zapytania z listingu 5.2.

	id_photo	*title*	*keyword*
1	1	Zachód słońca nad Wisłą	podróże
2	1	Zachód słońca nad Wisłą	natura
3	2	Pocztówka z Chorwacji	podróże
4	2	Pocztówka z Chorwacji	miasta
5	3	Mój ogród	rośliny
6	4	Konferencja pasjonatów modelarstwa	konferencje
7	5	Wystawa roślin Bonzai	konferencje
8	5	Wystawa roślin Bonzai	rośliny
9	6	Wspomnienia z podróży	podróże

Źródło: opracowanie własne

W złączeniach warunek nie zawsze musi być oparty o znak równości. Listing 5.3. przedstawia przykład utworzenia złączenia ze znakiem innym niż znak równości. Fragment wyniku został przedstawiony w tabeli 1.3. (oryginalnie zapytanie zwróciło 24 wiersze).

Listing 5.3. Przykład konstrukcji złączenia typu `OUTER JOIN`

```
SELECT Users.id_user, login, id_photo, title
FROM Users, Photos
WHERE Users.id_user *= Photos.id_user
```

Źródło: opracowanie własne

Tabela 5.3. Fragment wyniku zapytania z listingu 5.3.

	id_user	*login*	*id_photo*	*title*
1	1	mkowalski	1	Zachód słońca nad Wisłą
2	1	mkowalski	12	W ciemności – nowy firm A.Holland
3	2	anowicka	2	Pocztówka z Chorwacji
4	3	nborowiec	3	Mój ogród
5	3	nborowiec	5	Wystawa roślin Bonzai
6	4	pnowak	4	Konferencja pasjonatów modelarstwa
7	5	inowak	NULL	NULL
8	6	mwojcik	6	Wspomnienia z podróży

Źródło: opracowanie własne

Cechą charakterystyczną tego złączenia jest operator `*` który, umieszczony przed lub po znaku równości, określa charakter „nierównego” złączenia. Złączenie takie jest tożsame z omówionym dalej złączeniem typu `OUTER JOIN`. Jego podstawą jest nierówność zachodząca pomiędzy kluczami. Rezultatem (tabela 5.3) operacji jest wyświetlenie wszystkich rekordów z tabeli występującej po stronie operatora `*` oraz tylko odpowiadających im rekordów z drugiej tabeli. Tabela 5.3. przedstawia więc

wszystkich użytkowników, także tych, którzy nie dodali żadnego zdjęcia. Dla tych, którzy zdjęcia dodali, dodatkowo, wyświetlane są dane zdjęcia, a dla pozostałych – wartości `NULL`. Warto zauważyć, że zapytanie zwróciło 24 wiersze, podczas gdy analogiczne zapytanie oparte o równozłączenie – jedynie 20. Te 4 wiersze przedstawiają użytkowników, którzy zdjęć jeszcze nie dodawali.

Konstruując zapytania przedstawione w tym podrozdzialne sposób należy pamiętać, że standard ten nie jest polecany, ponieważ jest przestarzały. Dlatego też zaleca się korzystanie ze składni `OUTER JOIN` (Turley, Wood, 2009; MSDN, Using Outer Joins).

5.3. ZŁĄCZENIA PO KLAUZULI FROM (JOIN)

Najczęściej stosowanymi w praktyce typami złączeń są te zawierające klauzulę JOIN. Klauzula ta umieszczana jest po klauzuli `FROM`.

Istnieje kilka rodzajów złączeń typu JOIN. Są to: `INNER JOIN`, `OUTER JOIN` oraz Non-Equijoins (MSDN, Using Joins).

Klauzula INNER JOIN

Jako pierwsze opisane będą złączenia działające analogicznie jak te opisane w poprzednim rozdziale jako złączenia konstruowane po klauzuli `WHERE` z użyciem znaku równości. Do ich konstrukcji używa się klauzuli `INNER JOIN`. Klauzula ta oddziela dwie tabele umieszczone po klauzuli `FROM`. Bezpośrednio po nich umieszcza się warunek złączenia po słowie kluczowym `ON`.

Przykład zapytania zwracającego te same wyniki co zapytanie z listingu 5.1. przedstawia listing 5.4. Podobnie przykład umieszczony na listingu 5.5. przedstawia zapytanie, którego wynik pokrywa się z wynikiem zapytania z listingu 5.2.

Listing 5.4. Przykład konstrukcji złączenia dwóch tabel

```
SELECT Users.id_user, login, id_photo, title
FROM Users INNER JOIN Photos
ON Users.id_user=Photos.id_user
```

Źródło: opracowanie własne

Listing 5.5. Przykład konstrukcji złączenia trzech tabel

```
SELECT Photos.id_photo, title, keyword
FROM Photos INNER JOIN Keyword_to_photos
ON Photos.id_photo=Keyword_to_photos.id_photo
INNER JOIN Keywords
ON Keywords.id_keyword=Keyword_to_photos.id_keyword
```

Źródło: opracowanie własne

W zapytaniach z listingów 5.4. i 5.5. można pominąć słowo kluczowe `INNER`. Sama klauzula `JOIN` zachowuje się w ten sam sposób, co `INNER JOIN`.

Klauzula OUTER JOIN

Klauzula `OUTER JOIN` jest wykorzystywana do tworzenia złączeń w zapytaniach, których wynikiem mają być wszystkie wiersze z jednej tabeli i odpowiadające im wiersze z drugiej. Różnica między `OUTER JOIN` a `INNER JOIN` jest więc taka, że `INNER JOIN` zwraca rekordy spełniające tzw. równozłączenie, których zależność między kluczami jest oznaczana znakiem równości, podczas gdy `OUTER JOIN` może być traktowana jako zależność lewo- lub prawostronna (w zależności od kolejności umieszczenia tabel po klauzuli `FROM`).

Przykładem jest zapytanie przedstawione na listingu 5.6. Wynikiem są wszystkie rekordy z tabeli *Users* a dla tych, które mają swój odpowiednik w tabeli *Photos* są wyświetlane dodatkowe informacje (opisujące dane zdjęcia dodanego przez konkretnych użytkowników). Wynik jest analogiczny dla zapytania z listingu 5.3. ze złączeniem po słowie kluczowym `WHERE`.

Listing 5.6. Przykład konstrukcji złączenia typu `OUTER JOIN`

```
SELECT Users.id_user, login, id_photo, title
FROM Users LEFT OUTER JOIN Photos
ON Users.id_user=Photos.id_user
```

Źródło: opracowanie własne

Zapytania typu `OUTER JOIN` nie traktują jednakowo tabel, dlatego kolejność ich umieszczenia ma znaczenie. Zapytanie z listingu 5.6., na którym kolejność tabel byłaby odwrócona, zwróciłoby inny wynik. Zapytanie i jego wynik są przedstawione odpowiednio na listingu 5.7. i tabeli 5.4.

Listing 5.7. Przykład konstrukcji złączenia typu `OUTER JOIN`

```
SELECT Users.id_user, login, id_photo, title
FROM Photos LEFT OUTER JOIN Users
ON Users.id_user=Photos.id_user
```

Źródło: opracowanie własne

Warto zauważyć, że wyniki przedstawione w tabeli 5.4 są identyczne jak te będące wynikiem zapytania opartego na `INNER JOIN`. Dzieje się tak dlatego, że w przypadku zdjęć (tabela *Photos*) w bazie danych nie ma takich, do których nie jest przypisany żaden użytkownik.

Tabela 5.4. Wynik zapytania z listingu 5.7.

	id_user	***login***	***id_photo***	***title***
1	1	mkowalski	1	Zachód słońca nad Wisłą
2	2	anowicka	2	Pocztówka z Chorwacji
3	3	nborowiec	3	Mój ogród
4	4	pnowak	4	Konferencja pasjonatów modelarstwa
5	3	nborowiec	5	Wystawa roślin Bonzai
6	6	mwojcik	6	Wspomnienia z podróży

Źródło: opracowanie własne

Wpływ na wynik zapytania ma nie tylko kolejność umieszczania tabel po klauzuli `FROM`, ale również charakter samego złączenia. Można użyć klauzuli `LEFT OUTER JOIN` wyświetlającej wszystkie rekordy umieszczone w tabeli po lewej stronie złączenia lub też klauzuli `RIGHT OUTER JOIN`, która analogicznie traktuje prawą stronę złączenia. Przykład z listingu 5.6. z użyciem klauzuli `RIGHT OUTER JOIN` przedstawia listing 5.8.

Listing 5.8. Przykład konstrukcji złączenia typu `RIGHT OUTER JOIN`

```
SELECT Users.id_user, login, id_photo, title
FROM Photos RIGHT OUTER JOIN Users
ON Users.id_user=Photos.id_user
```

Źródło: opracowanie własne

Nierównozłączenia

Poza złączeniami opartymi o znak równości istnieje możliwość konstruowania złączeń z innymi znakami, takimi jak różne (<>), mniejsze (<), mniejsze-równe (<=),

większe (>), większe-równe (>=) (Turley, Wood, 2009). Przykład z listingu 5.9. przedstawia przykład użycia operatora mniejszości (<). Wynik przedstawiony został w tabeli 5.5.

Listing 5.9. Przykład wykorzystania operatora < w złączeniach

```
SELECT Photos.id_photo, title, date
FROM Photos INNER JOIN Photos_of_month
ON Photos.id_photo=Photos_of_month.id_photo
AND Photos_of_month.date<'2012-02-02'
```

Źródło: opracowanie własne

Tabela 5.5. Wynik zapytania z listingu 5.9.

	id_photo	***title***	***date***
1	1	Zachód słońca nad Wisłą	2011-12-01
2	8	Wspomnienia z podróży cd	2012-01-01
3	5	Wystawa roślin Bonzai	2012-02-01

Źródło: opracowanie własne

Warunek `Photos_of_month.date<'2012-02-02'` z listingu 5.9. może zostać także umieszczony po klauzuli `WHERE` (listing 5.10.). Obie wersje są poprawne i zwracają jednakowe wyniki.

Listing 5.10. Zapytanie z listingu 5.9. zapisane z użyciem klauzuli `WHERE`

```
SELECT Photos.id_photo, title, date
FROM Photos INNER JOIN Photos_of_month
ON Photos.id_photo=Photos_of_month.id_photo
WHERE Photos_of_month.date<'2012-02-02'
```

Źródło: opracowanie własne

Innym przykładem jest zapytanie umieszczone na listingu 5.11., które zwraca dane osób o takich samych imionach. Zapytanie oparte jest o złączenie dwóch kopii tej samej tabeli, którym nadane są różne aliasy. Złączenie jest utworzone ze znakiem różności (<>), aby uniknąć porównywania imion osób o tym samym identyfikatorze (np. osoby o identyfikatorze `1` z osobą o identyfikatorze `1`). Wynik zapytania przedstawia tabela 5.6.

Listing 5.11. Przykład wykorzystania operatora <> w złączeniu

```
SELECT DISTINCT u1.login, u1.name, u1.surname
FROM Users AS u1 INNER JOIN Users AS u2
ON u1.id_user<>u2.id_user
WHERE u1.name=u2.name
```

Źródło: opracowanie własne

Tabela 5.6. Wynik zapytania z listingu 5.11.

	Login	***name***	***surname***
1	amikulska	Anna	Mikulska
2	amucha	Anna	Mucha
3	anowicka	Anna	Nowicka

Źródło: opracowanie własne

5.4. DODATKOWE OPERACJE OPARTE O KLAUZULĘ JOIN

Istnieją dodatkowe rodzaje złączeń, które można konstruować z wykorzystaniem klauzuli `JOIN`. Są one nietypowe i rzadko używane, jednak w pewnych sytuacjach mogą okazać się przydatne. Operacje te to `FULL JOIN` oraz `CROSS JOIN` (Turley, Wood, 2009).

Pierwsza z nich (`FULL JOIN`) zwraca wynik typowy dla złączenia `OUTER JOIN` z wyłączeniem faworyzowania którejkolwiek z kolumn. Otrzymane w zapytaniu wyniki są niedopasowanymi z obu stron wartościami. Kolejność umieszenia tabel po klauzuli `FROM` nie ma tutaj znaczenia. Przykład zastosowania tego typu złączenia przedstawia listing 5.12. oraz tabela 5.7.

Listing 5.12. Przykład wykorzystania złączenia `FULL OUTER JOIN`

```
SELECT Photos.id_photo, title, date
FROM Photos FULL OUTER JOIN Photos_of_month
ON Photos.id_photo=Photos_of_month.id_photo
```

Źródło: opracowanie własne

Tabela 5.7. Fragment wyniku zapytania z listingu 5.12.

	id_photo	title	date
1	1	Zachód słońca nad Wisłą	2011-12-01
2	2	Pocztówka z Chorwacji	NULL
3	3	Mój ogród	NULL
4	4	Konferencja pasjonatów modelarstwa	NULL
5	5	Wystawa roślin Bonzai	2012-02-01
6	6	Wspomnienia z podróży	NULL
7	7	Wspomnienia z podróży cd	NULL

Źródło: opracowanie własne

Drugim typem złączenia jest `CROSS JOIN`, które zwraca iloczyn kartezjański rekordów ze złączonych tabel. Zapytanie zwróci więc wszystkie możliwe kombinacje wierszy w wybranych tabelach bez uwzględniania zgodnych wartości kluczy. Wynikiem takiego złączenia jest zwykle duża liczba wierszy, szczególnie jeśli łączonych jest wiele tabel. Warto zauważyć, że składnia złączenia nie przewiduje słowa kluczowego `ON`. Przykład przedstawia listing 5.13. oraz tabela 5.8.

Listing 5.13. Przykład wykorzystania złączenia `CROSS JOIN`

```
SELECT Photos.id_photo, title, date
FROM Photos CROSS JOIN Photos_of_month
ORDER BY Photos.id_photo
```

Źródło: opracowanie własne

Tabela 5.8. Fragment wyniku zapytania z listingu 5.13.

	id_photo	Title	date
1	1	Zachód słońca nad Wisłą	2011-12-01
2	1	Zachód słońca nad Wisłą	2012-01-01
3	1	Zachód słońca nad Wisłą	2012-02-01
4	1	Zachód słońca nad Wisłą	2012-03-01
5	1	Zachód słońca nad Wisłą	2012-04-01
6	2	Pocztówka z Chorwacji	2011-12-01
7	2	Pocztówka z Chorwacji	2012-01-01
8	2	Pocztówka z Chorwacji	2012-02-01
9	2	Pocztówka z Chorwacji	2012-03-01
10	2	Pocztówka z Chorwacji	2012-04-01
11	3	Mój ogród	2011-12-01
12	3	Mój ogród	2012-01-01

Źródło: opracowanie własne

Alternatywnym sposobem otrzymania takiego wyniku jest zapytanie utworzone z pominięciem warunku złączenia (listing 5.14.). Określone tabele są po prostu umieszczone po klauzuli `FROM` z pominięciem warunku obejmującego klucze.

Listing 5.14. Przykład wykorzystania złączenia `CROSS JOIN`

```
SELECT Photos.id_photo, title, date
FROM Photos, Photos_of_month
ORDER BY Photos.id_photo
```

Źródło: opracowanie własne

5.5. ZŁĄCZENIA TYPU UNION

Ważną możliwość w SQL stanowią zapytania z użyciem klauzuli `UNION`. Wyrażenie to pozwala na łączenie ze sobą rezultatów dwóch zapytań. Wyniki łączone są warstwowo w taki sposób, iż najpierw przedstawiane są rekordy z pierwszego zapytania, następnie dla drugiego. Nie można jednak w szybki sposób stwierdzić, gdzie kończy się wynik pierwszego zapytania a gdzie rozpoczyna się rezultat zwrócony przez kolejne.

Przykład ilustruje listing 5.15. oraz tabela 5.9. Zapytanie to jest sumą zbiorów będących wynikiem dwóch zapytań: użytkowników, którzy dodali komentarze do zdjęć w roku 2012 oraz użytkowników, którzy dodali zdjęcia w roku 2011. Oba zapytania zwracają te same kolumny w tej samej kolejności, co jest warunkiem poprawnego działania. Daje to też pewność otrzymania sensownych wyników.

Listing 5.15. Przykład wykorzystania operatora `UNION`

```
SELECT Users.id_user, login, name, surname
FROM Users INNER JOIN Comments
ON Users.id_user=Comments.id_user
WHERE YEAR(comment_create_date)='2012'
UNION
SELECT Users.id_user, login, name, surname
FROM Users INNER JOIN Photos
ON Users.id_user=Photos.id_user
WHERE YEAR(photo_create_date)='2011'
```

Źródło: opracowanie własne

Tabela 5.9. Wynik zapytania z listingu 5.15.

	id_user	***Login***	***name***	***surname***
1	1	Mkowalski	Maciej	Kowalski
2	2	Anowicka	Anna	Nowicka
3	3	Nborowiec	Natalia	Borowiec
4	5	Inowak	Iwona	Nowak
5	6	Mwojcik	Magdalena	Wójcik
6	8	Mkucharczyk	Maria	Kucharczyk
7	9	Amikulska	Anna	Mikulska
8	10	Kdebski	Karol	Dębski
9	11	Akot	Artur	Kot
10	14	Bjakubiak	Beata	Jakubiak
11	15	Akepa	Agnieszka	Kępa
12	16	Mpiatek	Michał	Piątek

Źródło: opracowanie własne

W zapytaniach zawierających klauzulę `UNION` ważne jest, aby oba zapytania zwracały tę samą liczbę kolumn o kompatybilnych typach danych. Najczęściej zapytania tego typu stosuje się dla kolumn o tych samych nazwach oraz typach danych. Jeśli nazwy są różne, nazwą kolumny (kolumn) wynikowej będzie nazwa kolumny (kolumn) z drugiego zapytania.

Przydatnym uzupełnieniem może być informacja o tym, z której tabeli pochodzi dany rekord. Przykład takiego rozwiązania przedstawia listing 5.16., gdzie dodana została dodatkowa kolumna `Source` określająca pochodzenie danego wiersza. Dla rekordów, które zostały zwrócone przez pierwsze zapytanie wyświetlona jest informacja '`Komentarze w 2012`', dla rekordów z drugiego zapytania natomiast: '`Zdjęcia w 2011`' (tabela 5.10.). Warto zauważyć, iż liczba wierszy zwróconych przez zapytanie z listingu 5.16. jest większa niż ta z listingu 5.15. Dzieje się tak dlatego, iż wyniki z listingu 5.15. zostały pozbawione powtarzających się wierszy (takich użytkowników, którzy występują w wynikach obu zapytań). Wynik z listingu 5.16. musi uwzględnić te rekordy oddzielnie ze względu na istnienie dodatkowej kolumny `Source`.

Listing 5.16. Przykład wykorzystania operatora `UNION`

```
SELECT Users.id_user, login, name, surname,
'Komentarze w 2011' AS source
FROM Users INNER JOIN Comments
ON Users.id_user=Comments.id_user
WHERE YEAR(comment_create_date)='2011'
UNION
```

```
SELECT Users.id_user, login, name, surname,
'Zdjęcia w 2012' AS source
FROM Users INNER JOIN Photos
ON Users.id_user=Photos.id_user
WHERE YEAR(photo_create_date)='2012'
```

Źródło: opracowanie własne

Tabela 5.10. Wynik zapytania z listingu 5.16.

	id_user	***login***	***name***	***surname***	***source***
1	1	mkowalski	Maciej	Kowalski	Zdjęcia w 2012
2	2	anowicka	Anna	Nowicka	Komentarze w 2011
3	2	anowicka	Anna	Nowicka	Zdjęcia w 2012
4	3	nborowiec	Natalia	Borowiec	Komentarze w 2011
5	3	nborowiec	Natalia	Borowiec	Zdjęcia w 2012
6	4	pnowak	Patryk	Nowak	Zdjęcia w 2012
7	6	mwojcik	Magdalena	Wójcik	Zdjęcia w 2012
8	7	mkowalczyk	Marek	Kowalczyk	Zdjęcia w 2012
9	8	mkucharczyk	Maria	Kucharczyk	Zdjęcia w 2012
10	12	jkimak	Joanna	Kimak	Zdjęcia w 2012
11	13	amucha	Anna	Mucha	Zdjęcia w 2012
12	14	bjakubiak	Beata	Jakubiak	Zdjęcia w 2012
13	15	akepa	Agnieszka	Kępa	Zdjęcia w 2012

Źródło: opracowanie własne

5.6. PODSUMOWANIE

Złączenia stanowią jedną z podstawowych operacji stosowanych w zapytaniach SQL. Zgodnie z aktualnie panującymi standardami złączenia powinny być, w miarę możliwości, konstruowane w oparciu o klauzulę `JOIN`. Pozwala ona na definiowanie różnego rodzaju złączeń, w szczególności typu `INNER` oraz `OUTER JOIN`. Przedstawione zostały przykłady złączeń zarówno w oparciu o dwie, jak i więcej tabel.

W rozdziale przedstawiono także rzadziej wykonywane operacje, takie jak `CROSS JOIN` i `FULL JOIN`. Wyjaśnione zostały również zasady korzystania z klauzuli `UNION`.

5.7. PYTANIA KONTROLNE

1. Wyjaśnij różnicę pomiędzy konstrukcją zapytań po klauzuli `WHERE` oraz po klauzuli `FROM`. Które z nich są bardziej przejrzyste?
2. Omów zasadę działania złączeń typu `OUTER JOIN`. Wyjaśnij zasady tworzenia takich złączeń.
3. Wyjaśnij zastosowanie złączeń typu `FULL JOIN` oraz `CROSS JOIN`.
   Omów zasady łączenia wyników zapytań.

6

Agregacja i grupowanie

Cel

W rozdziale omówiono dostępne funkcje agregujące oraz sposób ich wykorzystania. Przedstawiono zarówno przykłady zapytań korzystających jedynie z funkcji agregujących, jak również takich, które łączą agregację i inne konstrukcje, jak grupowanie. Obie te techniki umożliwiają dokonywanie analizy danych dzięki tworzeniu grup rekordów oraz wywoływania dla nich funkcji podsumowujących.

Plan

1. Funkcje agregujące.
2. Grupowanie.
3. Warunek `HAVING`
4. Dodatkowe opcje grupowania

6.1. WSTĘP

Agregacja i grupowanie rekordów to operacje niezwykle przydatne podczas przeprowadzania różnego rodzaju analiz danych. Analiza pojedynczych rekordów w tabeli jest czasochłonna i nie jest w stanie zapewnić odpowiedniego spojrzenia na dane. Taką właściwą perspektywę można uzyskać przy wykorzystaniu operacji agregacji i grupowania, które służą do przetwarzania dużych ilości danych i przedstawiania ich w określonych zestawieniach z uwzględnieniem żądanej perspektywy. To daje możliwość uzyskiwania statystyk, podsumowań i zestawień np. sprzedażowych czy biznesowych.

Funkcje agregujące w zapytaniach SQL spełniają więc zwykle rolę podsumowującą. Ich zadanie to najczęściej zwrócenie pojedynczej wartości podsumowującej (i agregującej) określoną w zapytaniu grupę rekordów. Istnieje też możliwość zastosowania funkcji agregujących do poszczególnych wierszy kolumn zapytania.

Zanim wykorzysta się funkcję agregującą, należy określić grupę rekordów, dla której funkcja ta będzie użyta. Dodatkowo warto rozważyć wykorzystanie grupowania, które w połączeniu z agregacją umożliwi wyświetlenie bardziej szczegółowych wyników z uwzględnieniem kontekstu analizy.

Jeśli dane są jednocześnie grupowane i agregowane, rezultat będzie listą niepowtarzalnych wartości pogrupowanych zgodnie z wartościami agregowanymi w każdej z grup. Jeżeli nie przewidziano grupowania, wynik zawierać będzie wartości funkcji agregujących dla poszczególnych wierszy.

6.2. AGREGACJA

Pola zwykle używane do agregacji to takie, które opisują charakterystykę rekordów – np.: kategoria, kolor, okres czasu (Turley, Wood, 2009). Pola te są wykorzystywane jako parametry funkcji agregujących. Większość funkcji agregujących przyjmuje jako parametr wartości numeryczne. Spotkać można jednak wyjątki takie jak np. COUNT, MIN czy MAX. Najprostszym sposobem wykorzystania funkcji agregującej jest użycie jej na liście pól umieszczonych po klauzuli SELECT.

Funkcje agregujące można podzielić na dwie grupy: proste funkcje agregujące oraz funkcje statystyczne. Zarówno jedne jak i drugie są dostępne w systemie MSSQL i wspierane przez standard T-SQL. Od wersji SQL Server 2005 użytkownik ma możliwość definiowania własnych funkcji agregujących. Funkcje te muszą być napisane w języku programowania wspieranym przez platformę .NET.

Proste funkcje agregujące

Do prostych funkcji agregujących zalicza się (MSDN, Aggregation Operations):

- `COUNT()` – określenie liczby wystąpień wartości nie będących `NULL` w zadanej kolumnie. W szczególności `COUNT(*)` można wykorzystać do zliczenia wszystkich wierszy zwróconych przez zapytanie. Funkcja zwraca wartości typu int.
- `COUNT_BIG()` – działanie analogiczne jak w przypadku funkcji `COUNT()` z tą różnicą, że funkcja `COUNT_BIG()` zwraca wartość typu `bigint`. Funkcji warto użyć w przypadku, gdy wynik zapytania zawiera więcej niż dwa miliardy wierszy.
- `SUM()` – określenie sumy wartości z zadanego zakresu z pominięciem wartości `NULL`. Funkcja zwraca typ danych zgodny z typem danych (numerycznych) analizowanej kolumny.
- `AVG()` – obliczenie wartości średniej w zadanym zakresie z pominięciem wartości `NULL`. Funkcja zwraca typ danych zgodny z typem danych numerycznych analizowanej kolumny.
- `MIN()` – określenie najmniejszej wartości w zakresie z pominięciem wartości `NULL`. Funkcja może być wywołana dla każdego sortowalnego typu danych (w tym daty i typów znakowych, gdzie wartość będzie określona na podstawie tablicy znaków ASCII).
- `MAX()` – określenie największej wartości w zakresie z pominięciem wartości `NULL`. Funkcja może być wywołana dla każdego sortowalnego typu danych.

Funkcje agregujące w najprostszej postaci są wywoływane dla pojedynczej kolumny i są jedynym wyrażeniem znajdującym się po klauzuli `SELECT` (Turley, Wood, 2009). Powoduje to, iż wynikiem zapytania jest jeden wiersz. Aby wynik był czytelniejszy można zdefiniować alias dla wynikowej kolumny. Kolejne przedstawiane w tym podrozdziale wyniki będą miały więc dla czytelności zdefiniowane aliasy.

Funkcja `COUNT()` jest jedną z najczęściej wykorzystywanych funkcji agregujących. Może ona działać na kolumnach o dowolnym typie danych. Typowym wykorzystaniem tej funkcji jest sprawdzenie liczby rekordów w tabeli. Przykład takiego zapytania przedstawia listing 6.1. Wynik zapytania przedstawiający liczbę rekordów utworzonych w roku 2012 w tabeli *Users* zaprezentowano w tabeli 6.1.

Listing 6.1. Przykład obliczenia liczby rekordów w tabeli przy użyciu funkcji `COUNT()`

```
SELECT COUNT(*) AS 'liczba użytkowników'
FROM Users
WHERE YEAR(user_create_date)='2012'
```

Źródło: opracowanie własne

Tabela 6.1. Wynik zapytania z listingu 6.1.

	liczba użytkowników
1	14

Źródło: opracowanie własne

Funkcję `COUNT()` można też wykorzystać do szybkiego sprawdzenia liczby użytkowników, którzy się logowali przynajmniej jeden raz (listing 6.2.). Wynik zapytania przedstawia tabela 6.2. Taki rezultat jest możliwy do otrzymania dzięki temu, że wartości `NULL` nie są przez funkcję `COUNT()` brane pod uwagę.

Korzystając z funkcji `COUNT()` należy zawsze zwracać uwagę na zapytanie, które uwzględnia potencjalnie występujące wartości `NULL`. Potwierdza to przykład przedstawiony na listingach 6.1. – 6.3. Funkcja `COUNT` reaguje inaczej, jeśli w wynikach pojawia się `NULL`. Jeżeli efekt ich pominięcia nie jest pożądany, zliczenia należy dokonywać po kolumnach o nałożonym warunku `NOT NULL`, np. po kluczu głównym. Funkcję `COUNT()` można też wywołać dla kolumny bez nałożonego warunku unikalności poprzedzonej słowem kluczowym `DISTINCT`. Takie rozwiązanie pozwoli zliczyć pojedyncze, niepowtarzalne wystąpienia poszczególnych wartości w kolumnie.

Listing 6.2. Przykład wykorzystania funkcji `COUNT()`

```
SELECT COUNT(last_login_success) AS 'liczba logujących się'
FROM Users
WHERE YEAR(user_create_date)='2012'
```

Źródło: opracowanie własne

Tabela 6.2. Wynik zapytania z listingu 6.2.

	liczba logujących się
1	10

Źródło: opracowanie własne

Funkcję `COUNT()` można też wywołać z użyciem słowa `DISTINCT`. Przykładem jest zapytanie z listingu 6.3., które zwraca liczbę zdjęć ocenionych już przynajmniej jeden raz. Informację tę można otrzymać poprzez zliczenie pojedynczych wystąpień wartości `id_photo` w tabeli `Rates`. Tabela ta zawiera dane o ocenach.

Listing 6.3. Przykład wykorzystania funkcji `COUNT()`

```
SELECT COUNT(DISTINCT id_photo) AS 'liczba ocenionych'
FROM Rates
```

Źródło: opracowanie własne

Kolejną popularną funkcją agregującą jest `SUM()`, która umożliwia dodawanie określonych w zapytaniu wartości. Wartości te pochodzą najczęściej z jednej kolumny. Listing 6.4. przedstawia zapytanie zwracające wartości sum ocen dla zdjęcia o zadanym id. Wyniki zapytania przedstawione są w tabeli 6.3.

Listing 6.4. Przykład wykorzystania funkcji `SUM ()`

```
SELECT SUM(rate) AS 'suma ocen'
FROM Rates
WHERE id_photo=1
```

Źródło: opracowanie własne

Tabela 6.3. Wynik zapytania z listingu 6.4.

	Suma ocen
1	14

Źródło: opracowanie własne

Kolejne listingi (6.5. – 6.6.) przedstawiają przykłady zastosowania funkcji `AVG()`, `MIN()` oraz `MAX()`. Przykład użycia funkcji `AVG()` przedstawiony na listingu 6.5. prezentuje jej zastosowanie dla wartości numerycznych, podczas gdy przedstawione na listingu 6.6. funkcje `MIN()` i `MAX()` działają na numerycznych typach daty i czasu. Wyniki zapytań przedstawione są odpowiednio w tabelach 6.4. i 6.5.

Zapytanie z listingu 6.5. zwraca średnią ocenę nadaną przez użytkownika o określonym loginie. Wynikiem jest wartość skalarna. Zapytanie prezentuje złączenie tabel *Users* (zawierającej dane użytkowników) i *Rates* (zawierającej oceny zdjęć).

Zapytanie z listingu 6.6. zwraca z kolei datę utworzenia pierwszego i ostatniego konta w tabeli *Users*. Wynikiem zapytania są jedynie daty, bez żadnych dodatkowych danych. Gdyby konieczne byłoby uzupełnienie tych danych o dodatkowe informacje, należałoby do zapytania dodać klauzulę grupującą.

Listing 6.5. Przykład wykorzystania funkcji `AVG()`

```
SELECT AVG(rate) AS 'średnia ocen'
FROM Rates INNER JOIN Users ON Rates.id_user=Users.id_user
WHERE login='mkowalski';
```

Źródło: opracowanie własne

Tabela 6.4. Wynik zapytania z listingu 6.5.

	średnia ocen
1	4

Źródło: opracowanie własne

Listing 6.6. Przykład wykorzystania funkcji `MIN()` i `MAX()`

```
SELECT MIN(user_create_date) AS 'data min',
MAX(user_create_date) AS 'data max'
FROM Users
```

Źródło: opracowanie własne

Tabela 6.5. Wynik zapytania z listingu 6.6.

	Data min	***Data max***
1	2011-03-10 12:10:00.000	2012-12-11 08:05:00.000

Źródło: opracowanie własne

Funkcje statystyczne

Do funkcji statystycznych zalicza się (Turley, Wood, 2009):

- `STDEV()` – obliczenie odchylenia standardowego w zakresie numerycznym dla wartości nie będących `NULL`. Niezależnie of typu analizowanej kolumny funkcja zwraca typ zmiennoprzecinkowy `float`.
- `STDEVP()` – obliczenie odchylenia standardowego dla populacji w zakresie numerycznym z uwzględnieniem wartości nie będących `NULL`. Niezależnie od typu analizowanej kolumny funkcja zwraca typ zmiennoprzecinkowy `float`.
- `VAR()` – obliczenie wariancji w zakresie numerycznym dla wartości nie będących `NULL`. Niezależnie od typu analizowanej kolumny funkcja zwraca typ zmiennoprzecinkowy `float`.
- `VARP()` – obliczenie wariancji dla populacji w zakresie numerycznym z uwzględnieniem wartości nie będących `NULL`. Niezależnie of typu analizowanej kolumny funkcja zwraca typ zmiennoprzecinkowy `float`.
- `CHECKSUM_AGG` – określenie sumy kontrolnej wartości w agregowanym zakresie. Funkcja wykorzystywana jest do porównania zgodności dwóch zakresów wartości (dwóch sum kontrolnych).

Najczęściej używanymi statystycznymi funkcjami agregującymi są te służące do wyznaczenia wariancji i odchylenia standardowego, które jest pierwiastkiem kwadratowym wartości wariancji.

Przykłady wykorzystania funkcji `STDEV()` oraz `VAR()` przedstawia listing 6.7. Wynikiem (tabela 6.6.) jest zestawienie odchylenia standardowego i wariancji dla wartości ocen w tabeli `Rates`. Dodatkowo umieszczono kolumnę zawierającą wartość pierwiastka kwadratowego z wartości wariancji. Jest on zgodny z wartością odchylenia standardowego.

Listing 6.7. Przykład wykorzystania funkcji statystycznych

```
SELECT STDEV(rate) AS 'odchylenie standardowe',
VAR(rate) AS 'wariancja',
SQRT(VAR(rate)) AS 'pierwiastek z war.'
FROM Rates
```

Źródło: opracowanie własne

Tabela 6.6. Wynik zapytania z listingu 6.7.

	odchylenie standardowe	***wariancja***	***pierwiastek z war.***
1	1,03762549441822	1,07666666666667	1,03762549441822

Źródło: opracowanie własne

6.3. GRUPOWANIE

Możliwości, jakie daje agregacja mogą być znacznie rozbudowane dzięki wykorzystaniu grupowania. Grupowanie umożliwia podział wyników zapytania pod względem wartości w określonych kolumnach. Grupowanie daje możliwość wykonywania sumarycznych obliczeń na każdej z grup (DeBetta i in, 2008).

Grupowanie to proces, który umożliwia uzyskanie podsumowania określonego zakresu rekordów na podstawie zadanych dopasowań i podobieństw. Zazwyczaj wszystkie wiersze w ramach grupy są agregowane w jednym rzędzie przy użyciu jednej lub wielu funkcji agregujących.

Kolumna lub kolumny, po których ma być przeprowadzone grupowanie, definiuje się po klauzuli `GROUP BY` umieszczanej po klauzulach `WHERE` oraz przed `ORDER BY` (listing 6.8). Aby poprawnie korzystać z możliwości grupowania należy przestrzegać kilku zasad (MSDN, Group by):

- Wszystkie kolumny, po których ma się odbyć grupowanie muszą zostać umieszczone po klauzuli `GROUP BY` lub użyte jako parametr funkcji agregującej.
- Wszystkie kolumny umieszczone po klauzuli `SELECT` muszą zostać uwzględnione jako dane grupowania.
- Inne kolumny (nie będące częścią listy grupowania) mogą być wykorzystane w konstrukcji warunków (po klauzuli `WHERE`) lub w parametrach sortowania (po klauzuli `ORDER BY`), jednak nie mogą znaleźć się one w wynikach zapytania.
- Grupowanie realizowane jest w taki sposób, że rekordy najpierw są wybierane zgodnie z zapytaniem (z uwzględnieniem złączeń i warunków) a następnie grupowane i agregowane.
- Grupowanie jednocześnie usuwa powtarzające się wartości w wyniku zapytania (działanie analogiczne jak w przypadku klauzuli `DISTINCT`).

Listing 6.8. Podstawowa struktura zapytania `SELECT` z grupowaniem

```
SELECT kolumna 1 [, kolumna 2] …[, kolumna n]
FROM tabela 1 [, tabela 2]… [, tabela n]
WHERE ograniczenie
GROUP BY kolumna 1 [, kolumna 2] …[, kolumna n]
ORDER BY sortowanie;
```

Źródło: opracowanie własne

Prostym przykładem operacji grupowania połączonej z agregacją jest ten z listingu 6.9. Wynik tego zapytania (tabela 6.7.) przedstawia listę użytkowników (`id_user` oraz `login`) oraz, dla każdego z nich oddzielnie, średnią ocen nadanych zdjęciom (bez podziału na poszczególne zdjęcia). Zapytanie oparte jest o dwie złączone tabele.

Listing 6.8. Przykład agregacji oraz grupowania

```
SELECT Users.id_user, login, AVG(rate) AS 'średnia ocen'
FROM Rates INNER JOIN Users ON Rates.id_user=Users.id_user
GROUP BY login, Users.id_user;
```

Źródło: opracowanie własne

Tabela 6.7. Wynik zapytania z listingu 6.8.

	id_user	***login***	***średnia ocen***
1	1	mkowalski	4
2	2	anowicka	4
3	4	pnowak	3
4	5	inowak	5
5	6	mwojcik	5
6	7	mkowalczyk	4
7	8	mkucharczyk	3
8	9	amikulska	3
9	10	kdebski	3
10	11	akot	2
11	12	jkimak	5
12	13	amucha	2
13	14	bjakubiak	5

Źródło: opracowanie własne

Wynik zapytania przedstawiony w tabeli 6.5. zawiera jedynie tych użytkowników, którzy ocenili przynajmniej jedno dowolne zdjęcie. Dane pozostałych użytkowników są pominięte na etapie złączenia. Uwzględnia ono jedynie te rekordy z tabeli *Users*, które mają odpowiednik w kluczu obcym (kolumna `id_user`) w tabeli *Rates*.

Wynik może zawierać kolumny `id_user` oraz `login`, ponieważ są one uwzględnione po klauzuli GROUP BY. Jednocześnie kolumna `rate` jest parametrem funkcji `AVG()`, więc nie musi być uwzględniana po `GROUP BY`. Grupy w tym przypadku składają się z pojedynczych użytkowników (wartości kolumny `id_user` i login). Zapytanie, dla każdej wartości z kolumny `id_user` i `login` (bez powtórzeń) zwraca średnią wartość oceny. Wynikiem są więc trzy kolumny.

Warto zauważyć, iż jeśli w zapytaniu nie umieści się części grupującej (listing 6.9.), zwrócony zostanie błąd (listing 6.10.). Aby zapytanie wykonało się poprawnie, w części grupującej muszą znaleźć się obie kolumny: `Users.id_user` oraz `login`.

Listing 6.9. Przykład agregacji oraz grupowania

```
SELECT Users.id_user, login, AVG(rate) AS 'średnia ocen'
FROM Rates INNER JOIN Users ON Rates.id_user=Users.id_user
```

Źródło: opracowanie własne

Listing 6.10. Wynik operacji z listingu 6.9.

```
Column 'Users.id_user' is invalid in the select list
because it is not contained in either an aggregate function or
the GROUP BY clause.
```

Źródło: opracowanie własne

Funkcje agregujące z powodzeniem można stosować także w zapytaniach korzystających ze złączeń typu `OUTER JOIN`. Przykład prezentuje listing 6.11., który zawiera liczbę ocen poszczególnych zdjęć, również takich, które nie zostały jeszcze ocenione. Jak widać w tabeli 6.8., zdjęcia które nie mają jeszcze ocen zostały uwzględnione w wyniku, a ich liczba ocen wynosi `0`. Złączenie `RIGHT OUTER JOIN` spowodowało wyświetlenie wszystkich rekordów wybranych z tabeli będącej po prawej stronie warunku złączenia (czyli z tabeli *Photos*). Istotne jest, aby wyświetlane `id_photo` było pobrane z tabeli *Photos*, ponieważ w tabeli *Rates* nie są zawarte wszystkie identyfikatory zdjęć, a jedynie te, które zostały ocenione. Ponadto w funkcji `COUNT` należy umieścić kolumnę należącą do tabeli *Rates*, aby możliwe było poprawne zliczenie ocen. Umieszczenie w funkcji `COUNT` operatora `*` zwróci błędne wyniki, ponieważ zliczenie nie będzie się odbywać po kolumnie z tabeli *Rates*.

Listing 6.11. Przykład agregacji w zapytaniu ze złączeniem `OUTER JOIN`

```
SELECT Photos.id_photo, title,
COUNT(id_rate) AS 'liczba ocen'
FROM Rates RIGHT OUTER JOIN Photos
ON Rates.id_photo=Photos.id_photo
GROUP BY Photos.id_photo, title
```

Źródło: opracowanie własne

Tabela 6.8. Fragment wyniku zapytania z listingu 6.11.

	id_photo	***title***	***liczba ocen***
1	1	Zachód słońca nad Wisłą	4
2	2	Pocztówka z Chorwacji	1
3	3	Mój ogród	2
4	4	Konferencja pasjonatów modelarstwa	1
5	5	Wystawa roślin Bonzai	0
6	6	Wspomnienia z podróży	2
7	7	Wspomnienia z podróży cd	2
8	8	Wspomnienia z podróży cd	3
9	9	Wspomnienia z podróży cd	1
10	10	Nasza córeczka	0
11	11	Nasz jamnik	0

Źródło: opracowanie własne

Listing 6.12. przedstawia zapytanie, w którym grupy tworzą dwie niezwiązane kolumny. Takie grupowanie po więcej niż jednej kolumnie tworzy grupy będące unikalną kombinacją wartości ze wszystkich kolumn i wyrażeń umieszczonych w grupowaniu. W pojedynczej grupie znajduje się więc jeden użytkownik i jedno zdjęcie. Średnia ocen wyświetlona w wyniku (tabela 6.9.) odnosi się więc do ocen konkretnych zdjęć dodanych przez poszczególnych użytkowników.

Listing 6.12. Przykład agregacji oraz grupowania

```
SELECT login, title, AVG(rate) AS 'średnia ocen'
FROM Rates INNER JOIN Users ON Rates.id_user=Users.id_user
INNER JOIN Photos ON Rates.id_photo=Photos.id_photo
GROUP BY login, title
ORDER BY login;
```

Źródło: opracowanie własne

Tabela 6.9. Fragment wyniku zapytania z listingu 6.12.

	login	***Title***	***średnia ocen***
1	akot	Zapiekanka z brokułami	2
2	amikulska	Mój ogród	3
3	amikulska	W ciemności – nowy firm A.Holland	4
4	amikulska	Warsztaty taneczne w Lublinie	4
5	amucha	Konferencja pasjonatów modelarstwa	2
6	anowicka	W ciemności – nowy firm A.Holland	5
7	anowicka	Wspomnienia z podróży	4
8	anowicka	Wspomnienia z podróży cd	5
9	bjakubiak	Kaktusy duże i małe	5
10	inowak	Wspomnienia z podróży cd	5
11	jkimak	Kaktusy duże i małe	5
12	kdebski	Wiosenna kolekcja firmy odzieżowej	3

Źródło: opracowanie własne

Warto zauważyć, iż bez użycia grupowania oraz funkcji agregującej trudno byłoby w szybki sposób przeprowadzić tego typu analizę. Dodatkowo przedstawioną analizę można uzupełnić dodając wywołanie funkcji agregujących (listing 6.13.). Wynik przedstawiony jest w tabeli 6.10.

Listing 6.13. Przykład agregacji oraz grupowania

```
SELECT login, title,
AVG(rate) as 'średnia ocen', SUM(rate) as 'suma ocen'
FROM Rates INNER JOIN Users ON Rates.id_user=Users.id_user
INNER JOIN Photos ON Rates.id_photo=Photos.id_photo
GROUP BY login, title
ORDER BY login;
```

Źródło: opracowanie własne

Tabela 6.10. Fragment wyniku zapytania z listingu 6.13.

	login	***title***	***średnia ocen***	***suma ocen***
1	akot	Zapiekanka z brokułami	2	2
2	amikulska	Mój ogród	3	3
3	amikulska	W ciemności – nowy firm A.Holland	4	4
4	amikulska	Warsztaty taneczne w Lublinie	4	4
5	amucha	Konferencja pasjonatów modelarstwa	2	2
6	anowicka	W ciemności – nowy firm A.Holland	5	5
7	anowicka	Wspomnienia z podróży	4	4
8	anowicka	Wspomnienia z podróży cd	5	10
9	bjakubiak	Kaktusy duże i małe	5	5
10	inowak	Wspomnienia z podróży cd	5	5
11	jkimak	Kaktusy duże i małe	5	5
12	kdebski	Wiosenna kolekcja firmy odzieżowej	3	3

Źródło: opracowanie własne

Istnieje możliwość zawężenia wyników zapytania zawierającego grupowanie i funkcje agregujące poprzez dodanie warunków opartych o funkcje grupujące. Taką możliwość daje klauzula `HAVING`.

6.4. WARUNEK HAVING

Filtrowanie rezultatów zapytania w oparciu o wyniki funkcji agregujących nie może być realizowane z wykorzystaniem klauzuli `WHERE`. Dzieje się tak dlatego, że grupowanie i agregacja są wywoływane już po wywołaniu złączeń oraz warunków umieszczonych po klauzuli `WHERE`. Aby wykonać filtrowanie po funkcji agregującej potrzebny jest więc dodatkowy mechanizm uruchamiany już po przeprowadzeniu grupowania i agregacji. Ten mechanizm to warunek `HAVING` (Nielsen i in 2009).

Mechanizm ten działa analogicznie jak mechanizm warunku `WHERE`. Różnica polega na kolejności ich wywołania. Warunek `WHERE` jest nakładany przed agregacją i grupowaniem, warunek HAVING natomiast – po. Ponadto warunek `HAVING` może działać jedynie na tych kolumnach i wyrażeniach agregujących, które są umieszczone po klauzuli `SELECT`.

Przykład ilustruje listing 6.14., prezentujący zapytanie wybierające zdjęcia, które mają zdefiniowane więcej niż jedno słowo kluczowe. Wynik przedstawia tabela 6.11.

Listing 6.14. Przykład wykorzystania klauzuli `HAVNG`

```
Select Photos.id_photo, title,
COUNT(Keyword_to_photos.id_keyword) AS 'liczba słów klucz.'
From Photos INNER JOIN Keyword_to_photos
ON Photos.id_photo=Keyword_to_photos.id_photo
INNER JOIN Keywords
ON Keywords.id_keyword=Keyword_to_photos.id_keyword
GROUP BY Photos.id_photo, title
HAVING COUNT(Keyword_to_photos.id_keyword)>1
```

Źródło: opracowanie własne

Tabela 6.11. Fragment wyniku zapytania z listingu 6.14.

	Id_photo	***Title***	***Liczba słów klucz.***
1	1	Zachód słońca nad Wisłą	2
2	2	Pocztówka z Chorwacji	2
3	5	Wystawa roślin Bonzai	2
4	6	Wspomnienia z podróży	2
5	8	Wspomnienia z podróży cd	2

Źródło: opracowanie własne

Przykład z listingu 6.14. można uzupełnić o dodatkowy warunek ograniczający wynik do zdjęć, które mają średnią ocenę wyższą niż 4. Tabela 6.12 przedstawia wynik zapytania, które przedstawione zostało na listingu 6.15.

Tabela 6.12. Fragment wyniku zapytania z listingu 6.15.

	Id_photo	***Title***	***Liczba słów klucz***	***Średnia ocen***
1	1	Zachód słońca nad Wisłą	8	4
2	2	Pocztówka z Chorwacji	2	4
3	3	Mój ogród	2	4
4	6	Wspomnienia z podróży	4	4
5	7	Wspomnienia z podróży cd	2	5
6	8	Wspomnienia z podróży cd	6	4

Źródło: opracowanie własne

Listing 6.15. Przykład wykorzystania klauzuli `HAVNG`

```
Select Photos.id_photo, title,
COUNT(Keyword_to_photos.id_keyword) AS 'liczba słów klucz',
AVG(Rates.rate) AS 'średnia ocen'
From Photos INNER JOIN Keyword_to_photos
ON Photos.id_photo=Keyword_to_photos.id_photo
INNER JOIN Keywords
ON Keywords.id_keyword=Keyword_to_photos.id_keyword
INNER JOIN Rates ON  Rates.id_photo=Photos.id_photo
GROUP BY Photos.id_photo, title
HAVING COUNT(Keyword_to_photos.id_keyword)>1
AND AVG(Rates.rate)>4
```

Źródło: opracowanie własne

Klauzula `HAVING` nie musi ograniczać się do wyrażeń opartych o funkcje agregujące. Może ona zostać wykorzystana do nałożenia dowolnych warunków złożonych z dowolnej kombinacji wyrażeń porównawczych i operatorów logicznych (podobnie jak w klauzuli `WHERE`). Należy jednak pamiętać o różnicy polegającej na kolejności uruchomienia i co za tym idzie na możliwych różnicach wydajności wykonania zapytania. Ponadto filtrowanie z wykorzystaniem klauzuli `HAVING` ograniczone musi być do kolumn i wyrażeń umieszczonych po klauzuli `SELECT` (Nielsen i in 2009). Dlatego też lepszą praktyką jest stosowanie `HAVING` tylko w przypadkach, które tego wymagają.

Listing 6.16. przedstawia dodatkowy przykład wykorzystujący zarówno klauzulę `WHERE` jak i `HAVING`. Wynik zapytania z listingu 6.16. przedstawia tabela 6.13. Zapytanie zwraca użytkowników, którzy dodali więcej niż 2 komentarze. Uwzględnia ono zdjęcia o słowie kluczowym *podróże*. Aby możliwe było obliczenie pojedynczych komentarzy konieczne jest umieszczenie słowa kluczowego `DISCTINT` w parametrze funkcji `COUNT()` przed kolumną `id_comment`. Konieczne jest również połączenie tabel: *Users*, *Comments*, *Photos* oraz *Keywords* i *Keyword_to_photos*.

Listing 6.16. Przykład wykorzystania klauzul `HAVNG` *i* `WHERE`

```
SELECT DISTINCT Users.id_user, login,
COUNT(DISTINCT Comments.id_comment) AS 'liczba komentarzy'
FROM Users INNER JOIN Comments
ON Users.id_user=Comments.id_user
INNER JOIN Photos ON Users.id_user=Photos.id_user
INNER JOIN Photos P ON Comments.id_photo=P.id_photo
INNER JOIN Keyword_to_photos
ON Keyword_to_photos.id_photo=Photos.id_photo
INNER JOIN Keywords
ON Keywords.id_keyword=Keyword_to_photos.id_keyword
WHERE keyword = 'podróże'
GROUP BY Users.id_user, login
HAVING COUNT(DISTINCT Comments.id_comment)>2
ORDER BY Users.id_user
```

Źródło: opracowanie własne

Tabela 6.13. Fragment wyniku zapytania z listingu 6.16.

	id_user	***Login***	***liczba komentarzy***
1	1	Mkowalski	3
2	6	Mwojcik	3

Źródło: opracowanie własne

6.5. DODATKOWE OPCJE GRUPOWANIA

Poza typowym grupowaniem oraz agregacją istnieje szereg dodatkowych opcji umożliwiających uzyskanie podsumowań i sum częściowych. Do takich opcji należą między innymi warunki `ROLLUP` i `CUBE` a także funkcja `GROUPING()` (Turley, Wood, 2009; MSDN, Using Group By With Rollup, Cube, and Grouping Sets).

Klauzula ROLLUP

Najprostszym z nich jest warunek `ROLLUP`, który używany jest do obliczenia sum częściowych oraz podsumowań opartych o pierwszą kolumnę występującą po klauzuli `GROUP BY` (Turley, Wood, 2009).

Listing 6.17. przestawia przykład wykorzystania warunku `ROLLUP`. Poza wynikami dla grup zapytanie zwraca również podsumowania dla kolejnych wartości w kolumnach `login` i `title` (tabela 6.14). Wartości `NULL` w wyniku wskazują, iż rzeczywiste wartości kolumny drugiej zostały zignorowane.

Listing 6.17. Przykład wykorzystania warunku `ROLLUP`

```
SELECT login, title, SUM(rate) as 'suma ocen'
FROM Rates INNER JOIN Users ON Rates.id_user=Users.id_user
INNER JOIN Photos ON Rates.id_photo=Photos.id_photo
where Photos.id_photo <4 AND Users.id_user<6
GROUP BY login, title
WITH ROLLUP
```

Źródło: opracowanie własne

Zapytanie z listingu 6.17. może również przyjmować formę przedstawioną na listingu 6.18. zgodną z SQL Server 2008. Rezultat obu zapytań będzie jednakowy.

Tabela 6.14. Fragment wyniku zapytania z listingu 6.17.

	login	***Title***	***suma ocen***
1	mkowalski	Pocztówka z Chorwacji	4
2	mkowalski	Zachód słońca nad Wisłą	3
3	mkowalski	NULL	7
4	pnowak	Zachód słońca nad Wisłą	3
5	pnowak	NULL	3
6	NULL	NULL	10

Źródło: opracowanie własne

Listing 6.18. Przykład wykorzystania warunku `GROUP BY ROLLUP`

```
SELECT login, title, SUM(rate) as 'suma ocen'
FROM Rates INNER JOIN Users ON Rates.id_user=Users.id_user
INNER JOIN Photos ON Rates.id_photo=Photos.id_photo
where Photos.id_photo <4 AND Users.id_user<6
GROUP BY ROLLUP (login, title)
```

Źródło: opracowanie własne

Klauzula CUBE

Warunek `CUBE` można uznać za rozszerzoną wersję operatora `ROLLUP` (Turley, Wood, 2009). Dzięki niemu uzyskać można dodatkowe podsumowania typu `ROLLUP` dla każdej kombinacji wartości kolumn lub wyrażeń tworzących grupy. W szczególności dotyczy to poszczególnych kolumn (analogicznie jak to miało miejsce z użyciem operatora `CUBE` dla pierwszej kolumny).

Listingi 6.19. i 6.20. przedstawiają sposoby wykorzystania warunku `CUBE` z użyciem klauzul `WITH CUBE` oraz `GROUP BY CUBE()`.

Listing 6.19. Przykład wykorzystania warunku `CUBE`

```
SELECT login, title, SUM(rate) as 'suma ocen'
FROM Rates INNER JOIN Users ON Rates.id_user=Users.id_user
INNER JOIN Photos ON Rates.id_photo=Photos.id_photo
WHERE Photos.id_photo <4 AND Users.id_user<6
GROUP BY login, title
WITH CUBE
```

Źródło: opracowanie własne

Listing 6.20. Przykład wykorzystania warunku `GROUP BY CUBE`

```
SELECT login, title, SUM(rate) as 'suma ocen'
FROM Rates INNER JOIN Users ON Rates.id_user=Users.id_user
INNER JOIN Photos ON Rates.id_photo=Photos.id_photo
where Photos.id_photo <4 AND Users.id_user<6
GROUP BY CUBE(login, title)
```

Źródło: opracowanie własne

Podobnie jak w przypadku `ROLLUP`, wartości `NULL` w wyniku oznaczają pominięcie konkretnych wartości kolumn. Zostają one „zwinięte” aby możliwe było wyświetlenie dodatkowego podsumowania dla określonych kolumn. Ostatni wiersz zawiera same wartości `NULL` co oznacza, iż jest to podsumowanie ogólne dla całego zbioru. Tabela 6.15. przechowuje wynik jednakowy dla obu zapytań.

Tabela 6.15. Fragment wyniku zapytania z listingów 6.19 i 6.20.

	login	***title***	***suma ocen***
1	mkowalski	Pocztówka z Chorwacji	4
2	NULL	Pocztówka z Chorwacji	4
3	mkowalski	Zachód słońca nad Wisłą	3
4	pnowak	Zachód słońca nad Wisłą	3
5	NULL	Zachód słońca nad Wisłą	6
6	NULL	NULL	10
7	mkowalski	NULL	7
8	pnowak	NULL	3

Źródło: opracowanie własne

Funkcja GROUPING()

Funkcja `GROUPING()` jest używana w skomplikowanych zapytaniach korzystających z klauzuli `ROLLUP`. Pozwala ona na uzyskanie wskaźnika wierszy będących podsumowaniem `ROLLUP` i tym samym na odróżnienie ich od wyników przedstawiających szczegółowe dane.

Funkcja zwraca jedno bitową wartość (`1` lub `0`) określającą czy wiersz powinien być uznany za `ROLLUP`, czy też nie. Ułatwia to uporządkowanie danych, a ponadto pozwala odseparować wyniki podsumowań dla rzeczywistych wartości `NULL` (których odróżnienie może być problematyczne, ponieważ wiersze `ROLLUP` również oznaczone są wartościami `NULL`). Takie rozwiązanie jest szczególnie przydatne, gdy dane są przekazywane i wykorzystywane przez aplikację. Przykład wykorzystania funkcji `GROUPING()` przedstawia listing 6.21.

Listing 6.21. Przykład wykorzystania funkcji `GROUPING`

```
SELECT login, title, SUM(rate) as 'suma ocen',
GROUPING(title) AS isRollup
FROM Rates INNER JOIN Users ON Rates.id_user=Users.id_user
INNER JOIN Photos ON Rates.id_photo=Photos.id_photo
where Photos.id_photo <4 AND Users.id_user<6
GROUP BY ROLLUP (login, title)
```

Źródło: opracowanie własne

Rezultat (tabela 6.16.) przedstawia dodatkową kolumnę `isRollup`. Każdy pojedynczy `ROLLUP` jest oznaczony wartością 1.

Tabela 6.16. Fragment wyniku zapytania z listingu 6.21.

	login	***title***	***suma ocen***	***isRollup***
1	mkowalski	Pocztówka z Chorwacji	4	0
2	mkowalski	Zachód słońca nad Wisłą	3	0
3	mkowalski	NULL	7	1
4	pnowak	Zachód słońca nad Wisłą	3	0
5	pnowak	NULL	3	1
6	NULL	NULL	10	1

Źródło: opracowanie własne

Można również, zamiast klauzuli `ROLLUP`, wykorzystać warunek `CUBE`. W takim wypadku pojawią się dodatkowe kolumny zawierające flagi `ROLLUP`.

6.6. PODSUMOWANIE

Rozdział przedstawia możliwości, jakie daje MSSQL w zakresie grupowania oraz podsumowywania danych. Omówione zostały podstawowe funkcje agregujące, takie jak `SUM`, `COUNT`, `MIN`, `MAX`. Przedstawiono także sposób ich wykorzystania w połączeniu z grupowaniem. Technika ta umożliwia spojrzenie na dane z innej, dalszej perspektywy co ułatwia analizę oraz czyni ją bardziej wydajną.

Ponadto omówiono dodatkowe warunki i funkcje ułatwiające pracę z pogrupowanymi danymi. Są to operatory `ROLLUP` i `CUBE` a także funkcja `GROUPING()`.

6.7. PYTANIA KONTROLNE

1. Podaj zasady konstrukcji zapytań wykorzystujących funkcje agregujące i grupowanie.
2. Scharakteryzuj podstawowy podział funkcji agregujących oraz przedstaw przykłady ich użycia.
3. Wyjaśnij różnicę między działaniem klauzul `WHERE` oraz `HAVING`.
4. Wyjaśnij zasady korzystania z klauzuli `HAVING`.

7

Podzapytania

Cel

Podzapytania rozumiane są jako zapytania zagnieżdżone, gdzie jedno zapytanie zawiera wewnątrz kolejne. Podzapytania umożliwiają konstruowanie rozbudowanych warunków i niejednokrotnie pozwalają na otrzymanie wyników, których nie udałoby się osiągnąć w żaden inny sposób. Celem rozdziału jest przedstawienie rodzajów podzapytań w języku SQL, zaprezentowanie przykładów ich konstrukcji oraz wyjaśnienie sposobu ich działania.

Plan

1. Podzapytania zwracające wyrażenia skalarne
2. Podzapytania zwracające zbiory rekordów
3. Podzapytania skorelowane

7.1. WSTĘP

Podzapytania mogą być tworzone oraz wykorzystywane na wiele różnych sposobów. W zależności od konstrukcji mogą one zwracać różne wartości, zarówno skalarne jak i wielo-kolumnowe o dużej liczbie wierszy. Rezultat podzapytania może być wykorzystany w warunku po słowie kluczowym `WHERE`, gdzie wynik może być np. przyrównany do innej wartości lub ich zbioru czy też wykorzystany jako parametr funkcji.

Warto zauważyć, iż jeśli istnieje możliwość zastąpienia podzapytania złączeniem – lepiej jest to zrobić ze względów wydajnościowych. Optymalizator zapytań znacznie lepiej działa w przypadku złączeń. W wielu przypadkach jednak podzapytania są bardziej skomplikowane i dają jedyną możliwość wykonania określonej operacji w jednym zadaniu.

Do typowych rodzajów podzapytań należą: wyrażenia skalarne, alternatywy dla operacji złączeń oraz podzapytania skorelowane. Zostaną one omówione w kolejnych podrozdziałach.

7.2. PODZAPYTANIA ZWRACAJĄCE WYRAŻENIA SKALARNE

Wyrażenia skalarne rozumiane są tu jako podzapytania zwracające tylko jedną kolumnę. Ich typowym przykładem jest wykorzystanie funkcji agregującej na wielu wierszach. Takie podzapytania umieścić można na liście wyrażeń po klauzuli `SELECT` lub w warunku WHERE (Turley, Wood, 2009). Podzapytania mogą być konstruowane w oparciu o kolumny tabel odpytywanych przez zapytanie na wyższym poziomie. Mogą też jednak odpytywać tabele niezwiązane z tabelami z zapytania wyższego poziomu.

Podzapytania proste, niebędące podzapytaniami skorelowanymi, są zwykle wykonywane jako pierwsze a ich wynik przekazywany jest jako fragment wyniku lub jako argument warunku zawężającego zbiór wyników.

Listing 7.1. przedstawia przykład prostego podzapytania umieszczonego po klauzuli `SELECT`. Wynikiem (tabela 7.1.) zapytania z listingu są dane użytkowników, a także dodatkowa kolumna (`Średnia ocen`), która dla każdego wiersza przyjmuje jednakową wartość ze względu na brak korelacji podzapytania z resztą zapytania. Wynik w kolumnie `Średnia ocen` nie jest więc w tym przypadku bezpośrednio związana z wyświetlanymi danymi użytkowników.

Listing 7.1. Przykład podzapytania umieszczonego po klauzuli `SELECT`

```
SELECT TOP 5 Users.id_user, login,
(SELECT AVG(rate) as 'średnia ocen' FROM Rates)
FROM Users;
```

Źródło: opracowanie własne

Tabela 7.1. Wynik zapytania z listingu 7.1.

	id_user	***login***	***średnia ocen***
1	1	mkowalski	4
2	2	anowicka	4
3	3	nborowiec	4
4	4	pnowak	4
5	5	inowak	4

Źródło: opracowanie własne

Można również wykorzystać podzapytanie jako element wyrażenia umieszczonego po klauzuli `SELECT` (listing 7.2.). Listing, dla każdej oceny zdjęcia, przedstawia różnicę pomiędzy nią a całkowitą średnią ocen wszystkich zdjęć w bazie danych. Wynik przedstawiony został w tabeli 7.2.

Listing 7.2. Przykład podzapytania umieszczonego jako wyrażenie po klauzuli `SELECT`

```
SELECT title, id_rate, rate,
ABS(rate -(SELECT AVG(rate) FROM Rates)) AS 'różnica'
FROM Rates INNER JOIN Photos
ON Rates.id_photo=Photos.id_photo
ORDER BY title
```

Źródło: opracowanie własne

Tabela 7.2. Fragment wyniku zapytania z listingu 7.2.

	title	id_rate	rate	różnica
1	Kaktusy duże i małe	16	5	1
2	Kaktusy duże i małe	17	5	1
3	Konferencja pasjonatów modelarstwa	24	2	2
4	Mój ogród	5	3	1
5	Mój ogród	6	5	1
6	Nowa kolekcja firmy odzieżowej	22	3	1
7	Nowa kolekcja firmy odzieżowej	23	3	1
8	Pocztówka z Chorwacji	4	4	0
9	W ciemności – nowy firm A.Holland	18	4	0
10	W ciemności – nowy firm A.Holland	19	5	1

Źródło: opracowanie własne

Listing 7.3. przedstawia przykład wykorzystania podzapytania w klauzuli `WHERE`. Zapytanie zwraca komentarze dodane przez tego użytkownika, który umieścił pierwsze zdjęcie w galerii. Identyfikator tego użytkownika jest wybierany na podstawie zdjęcia, które było dodane jako pierwsze (czyli ma najwcześniejszą datę). Data ta wybierana jest przez podzapytanie umieszczone po klauzuli `WHERE`, a następnie przekazywane jako warunek. Wynik podzapytania to '*2011-12-10 11:23:00*'. Identyczny wynik zwróciłoby zapytanie, którego warunek sformułowany byłby jako *WHERE photo_create_date= '2011-12-10 11:23:00'.* Wynik zapytania przedstawiony został w tabeli 7.3.

Listing 7.3. Przykład podzapytania umieszczonego po klauzuli `WHERE`

```
SELECT Users.id_user, login, comment
FROM Photos INNER JOIN Users
ON Photos.id_user=Users.id_user
INNER JOIN Comments ON Users.id_user=Comments.id_user
WHERE photo_create_date= (
SELECT min(photo_create_date)
FROM Photos)
```

Źródło: opracowanie własne

Tabela 7.3. Wynik zapytania z listingu 7.3.

	id_user	login	Comment
1	1	mkowalski	Wspaniałe miasto
2	1	mkowalski	Tłum turystów, ale i tak warto :)
3	1	mkowalski	Niesamowita świątynia

Źródło: opracowanie własne

Podzapytania zwracające wyrażenia skalarne mogą stanowić argument nie tylko po klauzuli `WHERE`, ale także po klauzuli `HAVING`. Przykład takiego podzapytania jest przedstawiony na listingu 7.4. Zapytanie zwraca listę użytkowników, którzy dodali najwięcej zdjęć. Podzapytanie wybiera maksymalną liczbę zdjęć dodanych przez jednego użytkownika (w tym wypadku – *4*). Liczba ta, wyznaczona przez podzapytanie, przekazywana jest do zapytania głównego, które na jej podstawie zwraca dane użytkowników o dodanej dokładnie takiej samej liczbie zdjęć (tabela 7.4). W przykładzie istnieje dokładnie jeden taki użytkownik.

Listing 7.4. Przykład podzapytania umieszczonego po klauzuli `HAVING`

```
SELECT Users.id_user, login, COUNT(*) AS 'liczba zdjęć'
FROM Users INNER JOIN Photos
ON Users.id_user=Photos.id_user
GROUP BY Users.id_user, login
HAVING COUNT(*) = (
SELECT TOP 1 COUNT(*)
FROM Photos GROUP BY id_user
ORDER BY COUNT(*) DESC)
```

Źródło: opracowanie własne

Tabela 7.4. Wynik zapytania z listingu 7.4.

	id_user	***login***	***liczba zdjęć***
1	6	mwojcik	4

Źródło: opracowanie własne

7.3. PODZAPYTANIA ZWRACAJĄCE ZBIORY REKORDÓW

Podzapytania mogą zwracać nie tylko pojedyncze wartości, ale także całe zbiory rekordów. Takie podzapytania stanowią nie tylko warunki zapytań, ale mogą także pełnić rolę złączeń. Istnieje kilka sposobów na wykorzystanie tego typu podzapytań (Lobel i in, 2009).

Przykład zapytania, które większość treści zawiera w podzapytaniu, ilustruje listing 7.5. Wynik zapytania zawiera tabela 7.5. Zapytanie wyświetla wszystkie rekordy, które zostały zwrócone przez podzapytanie. Wyniki podzapytania zostały oznaczone aliasem, aby możliwe było odwołanie się do nich przez zapytanie główne. W zapytaniu głównym, zamiast wyrażenia `S.*` można by było wskazać konkretne kolumny (`S.id_user`, `S.login`, `S.name` i/lub `S.surname`).

Listing 7.5. Przykład zapytania wybierającego całą treść podzapytania

```
SELECT S.*
FROM (
SELECT id_user, login, name, surname
FROM Users
GROUP BY id_user, login, name, surname
HAVING YEAR(MIN(user_create_date))='2011') AS S
```

Źródło: opracowanie własne

Tabela 7.5. Wynik zapytania z listingu 7.5.

	id_user	***login***	***name***	***Surname***
1	1	mkowalski	Maciej	Kowalski
2	2	anowicka	Anna	Nowicka

Źródło: opracowanie własne

Listing 7.6. Przykład zapytania korzystającego z operatora `IN`

```
SELECT *
FROM Photos
WHERE id_photo IN (
SELECT id_photo
FROM Rates
GROUP BY id_photo
HAVING AVG(rate)>4)
```

Źródło: opracowanie własne

Podzapytania często są łączone z zapytaniem głównym przy użyciu operatora `IN`. Pozwala on uwzględnić wynik podzapytania jako zbiór, który następnie może zostać uwzględniony jako dodatkowy warunek. Przykład przedstawia listing 7.6. Zapytanie uwzględnia zdjęcia, których średnia ocen jest powyżej 4. Podzapytanie zwraca identyfikatory takich zdjęć, które umożliwiają wyświetlenie pozostałych danych poszczególnych zdjęć. Wynik umieszczony został w tabeli 7.6. Warto zauważyć, że zapytanie prezentuje alternatywny sposób łączenia tabel. Ten sam wynik można również osiągnąć bez pomocy podzapytania, w oparciu o złączenie `INNER JOIN` tabel *Photos* i *Rates*.

Tabela 7.6. Wynik zapytania z listingu 7.6.

	id_user	***login***	***name***	***Surname***
1	1	mkowalski	Maciej	Kowalski
2	2	anowicka	Anna	Nowicka

Źródło: opracowanie własne

Podzapytania można zagnieżdżać. Takie zagnieżdżenie kilku podzapytań może z powodzeniem zastąpić tworzenie złączeń wielu tabel (Turley, Wood, 2009). Przykład z listingu 7.7. przedstawia przykład podwójnego zagnieżdżenia podzapytań. Zapytanie zwraca dane użytkowników, którzy logowali się już przynajmniej jeden raz i których zdjęcia były zdjęciami miesiąca w marcu i kwietniu 2012. Podzapytanie najbardziej zagnieżdżone zwraca identyfikatory zdjęć, na podstawie których zapytanie środkowego poziomu znajduje identyfikatory użytkowników. Te identyfikatory służą jako warunek zawężający listę wyświetlanych szczegółowych danych użytkowników. Wynik umieszczony został w tabeli 7.7.

Listing 7.7. Przykład zapytania korzystającego z operatora `IN`

```
SELECT id_user, login, name, surname
FROM Users
WHERE id_user IN (
SELECT id_user
FROM Photos
WHERE id_photo IN (
SELECT id_photo
FROM Photos_of_month
WHERE date BETWEEN '2012-03-01' AND '2012-04-01'))
AND last_login_success IS NOT NULL
```

Źródło: opracowanie własne

Tabela 7.7. Wynik zapytania z listingu 7.7.

	id_user	***login***	***name***	***surname***
1	16	mpiatek	Michał	Piątek

Źródło: opracowanie własne

Oddzielną grupę zastosowania podzapytań stanowią zapytania, które zwracać powinny tzw. *listę zanegowaną*. Przykładem takich zadań są np.: *wyświetlić tagi, które nie zawierają zdjęć* albo *wyświetlić użytkowników, którzy nigdy nie dodali żadnego komentarza*. Zapytania takie można skonstruować przy użyciu operatora `NOT IN`.

Przykład takiego użycia przedstawia listing 7.8. Zapytanie zwraca dane zdjęć, które nigdy nie zostały ocenione ani skomentowane. Podzapytanie, wykorzystując klauzulę `UNION`, zwraca sumę logiczną identyfikatorów zdjęć które są obecne w tabelach `Rates` lub `Comments`. Zbiór tych identyfikatorów jest przekazany do zapytania wyższego poziomu, gdzie za pomocą klauzuli `NOT IN` jest on eliminowany z wyników. Dzięki temu wynik zawiera jedynie te zdjęcia, których identyfikatory nie były zwrócone przez podzapytanie. Wynik zawiera tabela 7.8.

Listing 7.8. Przykład zapytania korzystającego z operatora `NOT IN`

```
SELECT id_photo, title, description, photo_create_date
FROM Photos
WHERE id_photo NOT IN (
SELECT DISTINCT id_photo
FROM Rates
UNION
SELECT DISTINCT id_photo
FROM Comments)
```

Źródło: opracowanie własne

Tabela 7.8. Wynik zapytania z listingu 7.8.

	id_photo	***Title***	***description***	***photo_create_date***
1	10	Nasza córeczka	NULL	2012-03-15 12:21:00.000
2	15	Najnowszy tablet firmy Asus	System operacyjny: Google Android	2012-02-13 15:09:00.000
3	17	Nowy kwiatek	Oto mój kącik kwiatowy	2012-03-11 13:12:00.000
4	19	Nasze plany Wakacyjne	Wyspy greckie na Morzu Jońskim	2012-01-15 13:25:00.000

Źródło: opracowanie własne

Drugi przykład zapytania z operatorem `NOT IN` jest przedstawiony na listingu 7.9. Wybiera ono użytkowników, których zdjęcia nigdy nie zostały zdjęciem miesiąca. Wynik zawarty jest w tabeli 7.9. Podzapytanie zwraca identyfikatory użytkowników, którzy dodali zdjęcia będące zdjęciem miesiąca. W podpytaniu musiało więc być dodane złączenie tabel *Photos* (w której były szukane identyfikatory) oraz `Photos_of_month` (zawierające dane zdjęć miesiąca). Przekazane do zapytania głównego identyfikatory zostały przy pomocy operatora `NOT IN` pominięte w wyniku.

Listing 7.9. Przykład zapytania korzystającego z operatora `NOT IN`

```
SELECT id_user, login, name, surname
FROM Users
WHERE id_user NOT IN (
SELECT id_user
FROM Photos_of_month INNER JOIN Photos
ON Photos.id_photo=Photos_of_month.id_photo)
```

Źródło: opracowanie własne

Tabela 7.9. Fragment wyniku zapytania z listingu 7.9.

	id_user	***login***	***Name***	***surname***
1	2	anowicka	Anna	Nowicka
2	4	pnowak	Patryk	Nowak
3	5	inowak	Iwona	Nowak
4	7	mkowalczyk	Marek	Kowalczyk
5	8	mkucharczyk	Maria	Kucharczyk
6	9	amikulska	Anna	Mikulska

Źródło: opracowanie własne

Dla realizacji podzapytań przydatną klauzulą może okazać się `TOP` (MSDN, TOP). Przykład jej wykorzystania przedstawiają listingi 7.12. oraz 7.13. Klauzula ta umożliwia ominięcie problemu podzapytania mogącego zwracać więcej niż jeden wiersz. Umożliwia ona wybór określonej ilości pierwszych wierszy zapytania. Ważne jest jednak, aby upewnić się, czy wiersze są posortowane zgodnie z zamierzeniami. W przeciwnym przypadku wyniki mogą okazać się błędne.

Klauzula `TOP` jest również przydatna do usunięcia powtarzających się wyników (np. w podzapytaniu). W niektórych przypadkach można więc traktować ją jako alternatywę dla klauzuli `DISTINCT`.

Zapytanie z listingu 7.10. zwracać powinno listę zdjęć, które należą do kategorii *podróże.* Podzapytanie w tym przypadku pełni rolę złączenia pomiędzy tabelami *Keywords* oraz *Keyword_to_photos*. Podczas uruchomienia zapytania zwracany jest błąd (listing 7.11.), ponieważ samo podzapytanie zwraca 7 wierszy (tabela 7.10.). Dzieje się tak dlatego, iż do tej właśnie kategorii należy więcej niż jedno zdjęcie , więc w tabeli *Keyword_to_photos* identyfikator słowa kluczowego *podróże* występuje wiele razy. Rozwiązaniem w tym przypadku jest wykorzystanie klauzuli `DISTINCT` lub `TOP` 1. Sposób z użyciem klauzuli `TOP` przedstawia listing 7.12., a wyniki – tabela 7.11.

Listing 7.10. Przykład zapytania, które może nie wykonać się poprawnie

```
SELECT Photos.id_photo, title
FROM Photos INNER JOIN Keyword_to_photos
On Photos.id_photo=Keyword_to_photos.id_photo
WHERE id_keyword= (SELECT Keywords.id_keyword
FROM Keyword_to_photos INNER JOIN Keywords
ON Keywords.id_keyword=Keyword_to_photos.id_keyword
WHERE keyword='podróże')
```

Źródło: opracowanie własne

Listing 7.11. Błąd podczas realizacji zapytania z listingu 7.10.

```
Subquery returned more than 1 value. This is not permitted
when the subquery follows =, !=, <, <= , >, >= or when the
subquery is used as an expression.
```

Źródło: opracowanie własne

Tabela 7.10. Wynik podzapytania z listingu 7.10.

	id_keyword
1	1
2	1
3	1
4	1
5	1
6	1
7	1

Źródło: opracowanie własne

Listing 7.12. Przykład poprawionej treści zapytania z listingu 7.10 z użyciem klauzuli `TOP`

```
SELECT Photos.id_photo, title
FROM Photos INNER JOIN Keyword_to_photos
On Photos.id_photo=Keyword_to_photos.id_photo
WHERE id_keyword=
(SELECT TOP 1 Keywords.id_keyword
FROM Keyword_to_photos INNER JOIN Keywords
ON Keywords.id_keyword=Keyword_to_photos.id_keyword
WHERE keyword='podróże')
```

Źródło: opracowanie własne

Tabela 7.11. Wynik zapytania z listingu 7.12.

	id_photo	***title***
1	1	Zachód słońca nad Wisłą
2	2	Pocztówka z Chorwacji
3	6	Wspomnienia z podróży
4	7	Wspomnienia z podróży cd
5	8	Wspomnienia z podróży cd
6	9	Wspomnienia z podróży cd
7	19	Nasze plany wakacyjne

Źródło: opracowanie własne

Drugim przykładem (listing 7.13.) jest zapytanie zwracające dane zdjęć, które były komentowane najczęściej. Podzapytanie musi zwrócić największą liczbę komentarzy, które zostały dodane do jednego zdjęcia. Aby było to możliwe wynik podzapytania należy posortować malejąco i zastosować klauzulę `TOP 1`. Tabela 7.12 przedstawia wynik zapytania z listingu 7.13.

Tabela 7.12. Wynik zapytania z listingu 7.13.

	id_photo	***title***	***liczba ocen***
1	1	Zachód słońca nad Wisłą	3
2	5	Wystawa roślin Bonzai	3
3	12	W ciemności – nowy firm A.Holland	3

Źródło: opracowanie własne

Listing 7.13. Przykład zapytania wykorzystującego klauzulę `TOP`

```
SELECT Photos.id_photo, title, COUNT(*) AS 'liczba ocen'
FROM Photos INNER JOIN Comments
ON Photos.id_photo=Comments.id_photo
GROUP BY Photos.id_photo, title
HAVING COUNT(*)=(
SELECT TOP 1 COUNT(*)
FROM Comments
GROUP BY id_photo)
```

Źródło: opracowanie własne

7.4. PODZAPYTANIA SKORELOWANE

Podzapytania skorelowane to takie, które bezpośrednio odnoszą się do zapytania głównego. Dzięki temu podzapytania takie mogą być wywoływane wiele razy w trakcie jednego przetwarzania całego zapytania (Nielsen i in 2009). Typowe zapytanie, które nie jest powiązane z zapytaniem nadrzędnym jest wywoływane tylko jeden raz a zapytanie nadrzędne korzysta jedynie z wyniku, który zwraca podzapytanie. W zapytaniach skorelowanych mechanizm jest inny – podzapytanie może zostać wywołane wiele razy np. dla poszczególnych wartości wiersza określonej kolumny. Działa więc ono na zasadzie pętli. Jest ona przekazywana do podzapytania przy użyciu aliasu, który umożliwia odwołanie się podzapytania do określonej kolumny wykorzystywanej w zapytaniu głównym.

Przykładem zadania wymagającego zastosowanie podzapytania skorelowanego jest informacja o tym kto i kiedy dodał pierwszy komentarz do zdjęć oznaczonych poszczególnymi słowami kluczowymi. Zapytanie to przedstawia listing 7.14.

Zapytanie wydaje się być dość złożone. Jego zadaniem jest wyświetlenie wszystkich słów kluczowych, które zostały wykorzystane do oznaczenia zdjęć. Dla każdego z nich wyświetlona jest dodatkowo informacja o dacie dodania pierwszego komentarza do dowolnego zdjęcia oznaczonego danym słowem kluczowym. Dodatkowo umieszczone są identyfikatory i loginy użytkowników, którzy dodali te pierwsze komentarze. Konieczne jest więc skonstruowanie podzapytania skorelowanego, które dla każdego słowa kluczowego wyszuka datę pierwszego dodanego komentarza. Data ta, przekazana do zapytania głównego, pozwoli na znalezienie użytkownika, który ten komentarz dodał. Korelacja, czyli połączenie zapytania głównego z podzapytaniem przeprowadzona jest po identyfikatorze słowa kluczowego z tabeli *Keyword_to_photos* oznaczonej aliasem `k`. Połączenie takie, z wykorzystaniem aliasu oznacza, że podzapytanie wywołane zostanie oddzielnie dla każdego słowa kluczowego. Wynik zapytania przedstawia tabela 7.13.

Listing 7.14. Przykład zapytania skorelowanego

```
SELECT k.id_keyword, keyword,
Comments.id_user, login, comment_create_date
FROM Keyword_to_photos k INNER JOIN Photos
ON Photos.id_photo=k.id_photo
INNER JOIN Comments ON Comments.id_photo=Photos.id_photo
INNER JOIN Keywords ON Keywords.id_keyword=k.id_keyword
INNER JOIN Users ON Users.id_user=Comments.id_user
WHERE comment_create_date = (
SELECT MIN(comment_create_date)
FROM Comments INNER JOIN Photos
ON Photos.id_photo=Comments.id_photo
INNER JOIN Keyword_to_photos
ON Photos.id_photo=Keyword_to_photos.id_photo
WHERE k.id_keyword=Keyword_to_photos.id_keyword
GROUP BY Keyword_to_photos.id_keyword)
ORDER BY k.id_keyword
```

Źródło: opracowanie własne

Tabela 7.13. Wynik zapytania z listingu 7.14.

	id_keyword	***keyword***	***id_user***	***login***	***comment_create_date***
1	1	podróże	2	anowicka	2011-12-12
2	2	konferencje	14	bjakubiak	2012-01-23
3	7	rośliny	6	mwojcik	2012-02-22
4	8	zwierzęta	9	amikulska	2012-04-11
5	10	kulinaria	11	akot	2012-08-17
6	11	natura	2	anowicka	2011-12-12
7	12	miasta	1	mkowalski	2012-01-16

Źródło: opracowanie własne

Listing 7.15. przedstawia podobne zapytanie skorelowane zwracające dane użytkowników, którzy jako pierwsi dodali zdjęcia przypisane do poszczególnych słów kluczowych. Korelacja również przeprowadzona została po identyfikatorach słów kluczowych z tabel *Keyword_to_photos*. Aby korelacja była możliwa, do tabeli *Keyword_to_photos* w zapytaniu głównym dodano alias, który wykorzystuje podzapytanie. Podzapytania dla każdego słowa kluczowego zwraca datę pierwszego dodanego zdjęcia oznaczonego danym słowem kluczowym. Wynik zapytania przedstawia tabela 7.14.

Listing 7.15. Przykład zapytania skorelowanego

```
SELECT Keywords.id_keyword, keyword,
Photos.id_user, login, photo_create_date
FROM Keyword_to_photos k INNER JOIN Keywords
ON Keywords.id_keyword=k.id_keyword
INNER JOIN Photos ON Photos.id_photo=k.id_photo
INNER JOIN Users ON Users.id_user=Photos.id_user
WHERE photo_create_date = (
SELECT MIN(photo_create_date)
FROM Photos INNER JOIN Keyword_to_photos
ON Photos.id_photo=Keyword_to_photos.id_photo
WHERE k.id_keyword=Keyword_to_photos.id_keyword
GROUP BY Keyword_to_photos.id_keyword)
ORDER BY Keywords.id_keyword
```

Źródło: opracowanie własne

Tabela 7.14. Wynik zapytania z listingu 7.15.

	id_keyword	***keyword***	***id_user***	***login***	***photo_create_date***
1	1	podróże	1	mkowalski	2011-12-10
2	2	konferencje	15	akepa	2012-01-22
3	3	moda	13	amucha	2012-03-11
4	4	sprzęt	12	jkimak	2012-02-13
5	5	komputery	12	jkimak	2012-02-13
6	7	rośliny	3	nborowiec	2012-02-22
7	8	zwierzęta	8	mkucharczyk	2012-04-01
8	9	rodzina	7	mkowalczyk	2012-03-15
9	10	kulinaria	14	bjakubiak	2012-04-15
10	11	natura	1	mkowalski	2011-12-10
11	12	miasta	1	mkowalski	2012-01-11

Źródło: opracowanie własne

Wyjątkowym rodzajem podzapytań skorelowanych są te wykorzystujące funkcję `EXISTS` (MSDN, Exists). Funkcja ta sprawdza istnienie wartości wiersza zapytania zewnętrznego w podzapytaniu (Turley, Wood, 2009). Podzapytanie może być oparte o dowolne tabele i kolumny, nawet te niezwiązane z wyrażeniami w zapytaniu nadrzędnym. Funkcja ta pozwala na konstrukcję zapytań, których budowa byłaby trudna lub wręcz niemożliwa przy użyciu innych operacji.

Przykład zapytania opartego o klauzule `EXISTS` oraz `HAVING` przedstawia listing 7.16. Zapytanie zwraca informacje o użytkownikach, którzy dodali do zdjęć oceny o średniej poniżej 3. Wynik zapytania prezentuje tabela 7.15.

Listing 7.16. Przykład zapytania z klauzulą `EXISTS`

```
SELECT id_user, login
FROM Users
WHERE EXISTS(
SELECT AVG(rate) FROM Rates
WHERE Rates.id_user=Users.id_user
HAVING AVG(rate)<3)
```

Źródło: opracowanie własne

Tabela 7.15. Wynik zapytania z listingu 7.16.

	id_user	***login***
1	11	akot
2	13	amucha

Źródło: opracowanie własne

Drugim przykładem wykorzystania klauzuli `EXISTS` jest listing 7.17., przedstawiający zapytanie zwracające szczegóły zdjęć o słowach związanych z podróżami. Warto zwrócić uwagę, iż korelacja w tym zapytaniu jest przeprowadzana w oparciu o kolumnę *`id_photo`* oraz tabelę *Photos* opatrzoną aliasem wykorzystywanym w podzapytaniu. Wynik prezentuje tabela 7.16.

Listing 7.17. Przykład zapytania z klauzulą EXISTS

```
SELECT id_photo, title
FROM Photos p
WHERE EXISTS(
SELECT id_photo
FROM Keyword_to_photos INNER JOIN Keywords
ON Keyword_to_photos.id_keyword=Keywords.id_keyword
WHERE p.id_photo=Keyword_to_photos.id_photo
AND keyword IN ('podróże','natura','miasto'))
```

Źródło: opracowanie własne

Tabela 7.16. Wynik zapytania z listingu 7.17.

	id_photo	*title*
1	1	Zachód słońca nad Wisłą
2	2	Pocztówka z Chorwacji
3	6	Wspomnienia z podróży
4	7	Wspomnienia z podróży cd
5	8	Wspomnienia z podróży cd
6	9	Wspomnienia z podróży cd
7	19	Nasze plany wakacyjne

Źródło: opracowanie własne

Istnieje również klauzula `NOT EXISTS` (MSDN, Exists), której działanie jest odwrotne w stosunku do działania klauzuli `EXISTS`. Przykładem wykorzystania tej klauzuli jest zapytanie przedstawione na listingu 7.18. zwracające dane użytkowników, którzy nie dodali żadnego zdjęcia. Wynik zapytania z listingu 7.18. przedstawiony został w tabeli 7.17.

Listing 7.18. Przykład zapytania z klauzulą `NOT EXISTS`

```
SELECT id_user, login
FROM Users u
WHERE NOT EXISTS(
SELECT id_user
FROM Photos
WHERE u.id_user=id_user)
```

Źródło: opracowanie własne

Tabela 7.17. Wynik zapytania z listingu 7.18.

	id_user	*login*
1	5	inowak
2	9	amikulska
3	10	kdebski
4	11	akot

Źródło: opracowanie własne

7.5. PODSUMOWANIE

Przedstawione w tym rozdziale przykłady podzapytań ilustrują duży wachlarz zastosowań w praktyce. Wiele bardziej skomplikowanych zadań musi być realizowanych w oparciu o podzapytania. Dają one konstruktorowi sporą swobodę w analizie danych oraz tworzenia złożonych warunków.

Warto zauważyć, że podzapytania można dowolnie zagnieżdżać. Istnieje również możliwość skorzystania z podzapytań skorelowanych. Podczas korzystania z podzapytań należy pamiętać, iż podzapytanie najczęściej jest realizowane w pierwszej kolejności. Jego wynik jest następnie przekazywany do zapytania wyższego poziomu. Wyjątek stanowią jedynie podzapytania skorelowane, których powiązanie z zapytaniem na wyższym poziomie jest najczęściej cykliczne.

Podzapytania nie należą jednak do najwydajniejszych narzędzi bazy danych. Dzieje się tak dlatego, iż optymalizator zapytań nie jest w stanie analizować całego złożonego zapytania jednocześnie, dlatego podzapytania są analizowane oddzielnie.

7.6. PYTANIA KONTROLNE

1. Omów zastosowania podzapytań.
2. Wyjaśnij, w jaki sposób podzapytania mogą pełnić rolę alternatywy dla złączeń.
3. Wyjaśnij różnicę pomiędzy uruchamianiem podzapytań prostych i podzapytań skorelowanych.
4. Podaj przykład podzapytania zwracającego wartość skalarną.

8

Funkcje systemowe i zmienne

Cel

Rozdział przedstawia najczęściej używane funkcje oraz zmienne systemowe. Są one przydatne zarówno do tworzenia zapytań jak i do realizacji zadań administracyjnych. Przedstawione zostały sposoby korzystania z funkcji, ich zastosowanie w zapytaniach oraz zagnieżdżania.

Funkcje można pogrupować w kategorie: funkcje agregujące, zmienne konfiguracyjne, funkcje konwersji, kursory, funkcje daty i czasu, funkcje matematyczne, rankingowe, zmienne konfiguracyjne, statystyki systemowe, funkcje bezpieczeństwa i manipulacji systemem.

Plan

1. Grupa funkcji agregujących
2. Zmienne konfiguracyjne
3. Funkcje konwersji
4. Funkcje daty
5. Funkcje manipulacji ciągów znaków
6. Funkcje matematyczne i funkcje metadanych

8.1. WSTĘP

Język SQL, poza prostym wyborem danych, pozwala też na ich przetwarzanie. Taka manipulacja danymi umożliwia otrzymanie przydatnych, użytecznych wyników. Do najczęściej wykorzystywanych w praktyce operacji można zaliczyć przetwarzanie matematyczne, konwertowanie danych, łączenie oraz agregowanie danych. Wszystkie te operacje wykonywane są za pomocą funkcji. Ogólny podział funkcji przedstawiony został w tabeli 8.1.

Tabela 8.1. Grupy funkcji SQL i T-SQL

Kategoria	Przeznaczenie
Agregacja	Zwraca wartość skalarną reprezentującą agregację po określonym zakresie wartości
Zmienne konfiguracyjne	Zwracają informacje dotyczące przydatnych zmiennych środowiskowych SQL Server
Konwersja	Konwersja wartości jednego typu do innego, często stosowana także do zmiany formatowania danych takich jak: data, czas, czy wartości numeryczne
Kursor	Pętla mająca charakter proceduralny, przebiegająca iteracyjnie po wierszach
Data i czas	Przetwarzanie, formatowanie oraz manipulacja danymi daty i czasu
Matematyczne	Specjalistyczne operacje matematyczne z zakresu algebry, trygonometrii, statystyki, oszacowania, operacji finansowych
Rankingowe	Przedstawianie posortowanych, wybranych wartości
Bezpieczeństwa	Ustawienia uprawnień użytkowników SQL Server. Zarządzanie wyjątkami.
Manipulacja systemem	Parsowanie, zmiana oraz manipulacja zmiennymi
Systemowe	Zbiór funkcji o różnych zastosowaniach takich tak porównywanie wartości, sprawdzanie typu danych
Statystyki systemowe	Funkcje administracyjne używane do sprawdzania użyteczności systemu bazy danych oraz środowiska

Źródło: (Turley, Wood, 2009)

Funkcje systemowe są często używane w połączeniu z zapytaniami. Aby wywołać funkcję, jej nazwę umieszcza się na liście kolumn po słowie kluczowym SELECT. Argumentami funkcji umieszczonych w zapytaniu są zwykle nazwy kolumn (listing 8.1.). Ponadto funkcje można z powodzeniem zagnieżdżać. Można także w jednym zapytaniu umieszczać wiele funkcji po klauzuli SELECT oraz w podzapytaniach. Wynik zapytania z listingu 8.1 dotyczącego sformatowanej aktualnej daty przedstawia tabela 8.2.

Listing 8.1. Przykład użycia funkcji systemowych w zapytaniu

```
SELECT YEAR(GETDATE()) AS rok
```

Tabela 8.2. Wynik zapytania z listingu 8.1.

	rok
1	2012

Źródło: opracowanie własne

8.2. GRUPA FUNKCJI AGREGUJĄCYCH

Jedną z najistotniejszych grup funkcji są funkcje agregujące, które pozwalają grupować rekordy według określonego kryterium. Są one wykorzystywane do tworzenia raportów w oparciu o dane. Funkcje agregujące zwracają pojedynczą wartość skalarną odpowiadającą zadanej operacji. Zwracany typ jest zgodny z typem kolumny przekazanej do funkcji. Funkcje grupujące są bardzo często wykorzystywane w połączeniu z operacjami grupowania.

Wśród funkcji grupujących można wyróżnić (Nielsen i in, 2009):

- `AVG()` – wartość średnia. Parametrem funkcji jest zwykle jedna kolumna. W przypadku wystąpienia w kolumnie wartości `NULL`, są one pomijane.
- `COUNT()` – zliczenie pojedynczych wystąpień. Zliczenie jako parametr może przyjmować konkretną kolumnę lub znak `*` oznaczający zliczanie wszystkich wierszy, które zostały zwrócone przez zapytanie. Wartości `NULL` są pomijane.
- `MIN()` i `MAX()` – zwracają odpowiednio najmniejszą i największą wartość w zakresie wskazanym przez kolumnę. Warto zauważyć, iż wynik operacji będzie się różnić w zależności od typu danych. Np. dla typów znakowych minimalna i maksymalna wartość będzie określona zgodnie z kolejnością znaków ASCII.
- `SUM()` – suma wartości w kolumnie. Podobnie jak w przypadku funkcji `AVG`, funkcja `SUM` pomija wartości `NULL`.

Funkcje agregujące zostały szczegółowo opisane w rozdziale Agregacja i grupowanie.

8.3. ZMIENNE KONFIGURACYJNE

Zmienne konfiguracyjne nie są funkcjami, ale mogą być używane w podobny sposób. Każda zmienna zwraca wartość skalarną oznaczającą określony stan poszczególnych elementów środowiska bazy danych. Do przykładowych zmiennych konfiguracyjnych można zaliczyć (Turley, Wood, 2009):

- `@@ERROR` – zmienna zawierająca numer ostatniego błędu, jaki pojawił się podczas aktualnego połączenia. Jeśli nie pojawiły się żadne błędy, zmienna będzie przyjmować wartość `0`. Błędy są zwracane przez bazę danych podczas wystąpienia standardowych wyjątków. Można również obsłużyć zwracanie własnych błędów (tzw. `Custom errors`) z wykorzystaniem wyrażenia `RAISERROR`.
- `@@SERVICENAME` – zmienna przechowująca nazwę usługi Windows odpowiedzialną za aktualną instancję SQL Serwera.
- `@@TOTAL_ERRORS` – zmienna określająca całkowitą liczbę błędów, które wystąpiły od początku aktualnego połączenia z bazą danych.
- `@@TOTAL_READ` – zmienna przechowująca całkowitą liczbę operacji dyskowych przeprowadzonych od początku ustanowienia połączenia z bazą danych.
- `@@VERSION` – zmienna zawierająca kompletną informację o aktualnej instancji SQL Serwera.

Listing 8.2. przedstawia zapytanie zwracające wartość zmiennej @@TOTAL_READ. Wynik przedstawia tabela 8.3.

Listing 8.2. Przykład wykorzystania zmiennej `@@TOTAL_READ`

```
SELECT @@TOTAL_READ AS liczba_op
```

Źródło: opracowanie własne

Tabela 8.3. Wynik zapytania z listingu 8.2.

	liczba_op
1	1862

Źródło: opracowanie własne

Do obsługi błędów przydatna jest znajomość błędów systemowych. Wszystkie standardowe kody oraz opisy błędów są dostępne w perspektywie sys.message (listing 8.3.). Tabela 8.4. przedstawia kilka pierwszych wierszy z otrzymanego wyniku zapytania.

Listing 8.3. Zapytanie do perspektywy sys.message

```
SELECT * FROM sys.messages
```

Źródło: opracowanie własne

Tabela 8.4. Fragment wyniku zapytania z listingu 8.3.

message_id	language_id	severity	is_event_logged	text
21	1033	20	0	Warning: Fatal error %d occurred at %S_DATE. Note the error and time, and contact your system administrator.
101	1033	15	0	Query not allowed in Waitfor.
102	1033	15	0	Incorrect syntax near '%.*ls'.
103	1033	15	0	The %S_MSG that starts with '%.*ls' is too long. Maximum length is %d.
104	1033	15	0	ORDER BY items must appear in the select list if the statement contains a UNION, INTERSECT or EXCEPT operator.

Źródło: opracowanie własne

8.4. FUNKCJE KONWERSJI

Konwersję typów przeprowadza się przede wszystkim z wykorzystaniem dwóch funkcji: CAST() oraz CONVERT() (MSDN, Cast and Convert). Funkcje te mają różną składnię, choć ich działanie jest zbliżone. Warto zauważyć, iż funkcje te nie zaokrąglają wartości, nie starają się też odgadnąć intencji użytkownika. Jeśli określona wartość nie może zostać przekonwertowana, funkcja zwróci błąd. Przykładowo wartość 3,14 nie może zostać poprawnie skonwertowana na typ int.

Funkcja CAST() przyjmuje jeden argument, który zawiera zarówno wartość, jak i docelowy typ. Części te rozdzielone są słowem AS. Przykład wykorzystania funkcji przedstawia listing 8.4.

Listing 8.4. Przykład wykorzystania funkcji `CAST()`

```
Select CAST ('42' AS int)
```

Źródło: opracowanie własne

Funkcja CAST() pozwala też na określenie dodatkowych właściwości zwracanych wartości. Przykładowo, dla typu zmiennoprzecinkowego decimal można ustawić precyzję (całkowita liczba cyfr przed i po przecinku) oraz skalę (liczba cyfr po przecinku). Domyślnie typ decimal przyjmuje wartości precyzji oraz skali odpowiednio 18 i 0 co oznacza, że zwracana wartość jest całkowita, a jej część po przecinku zostaje odcięta. Przykład takiej operacji przedstawia listing 8.5.

Listing 8.5. Przykład wykorzystania funkcji `CAST()`

```
Select CAST ('42.42' AS decimal (6,1))
```

Źródło: opracowanie własne

Kolejny przykład bardziej złożonej operacji z wykorzystaniem funkcji CAST() przedstawia listing 8.6. Wyniki (tabela 8.5.) pokazują sposób działania funkcji CAST(), która, niezależnie od liczby cyfr pól w kolumnie id_user, zamieniła je na dwu-znakowe ciągi.

Listing 8.6. Przykład wykorzystania funkcji `CAST()`

```
SELECT CAST(id_user AS char(2))
+ '_' + login
+ '_'
+ e_mail AS uzytkownik
FROM gallery.dbo.Users
WHERE id_user IN (1,2,9,10,11)
```

Źródło: opracowanie własne

Tabela 8.5. Wynik zapytania z listingu 8.6.

	uzytkownik
1	1 _mkowalski_mkowalski@email.pl
2	2 _anowicka_anowicka@e.pl
3	9 _amikulska_amikulska@op.com
4	10_kdebski_kdebski@on.eu
5	11_akot_akot@kot.pl

Źródło: opracowanie własne

Aby z wyniku z tabeli 8.5. usunąć nadmiarowe spacje można w zapytaniu użyć funkcji LTRIM() (MSDN, Ltrim), która usuwa z wyrażenia zbędne białe znaki (listing 8.7. i tabela 8.6.). Funkcja ta będzie bardziej szczegółowo omówiona w dalszej części rozdziału.

Listing 8.7. Przykład wykorzystania funkcji `CAST()`

```
SELECT LTRIM(CAST(id_user AS char(2)))
+ '_' + login
+ '_'
+ e_mail AS uzytkownik
FROM gallery.dbo.Users
WHERE id_user IN (1,2,9,10,11)
```

Źródło: opracowanie własne

Tabela 8.6. Wynik zapytania z listingu 8.7.

	uzytkownik
1	1 _mkowalski_mkowalski@email.pl
2	2 _anowicka_anowicka@e.pl
3	9 _amikulska_amikulska@op.com
4	10_kdebski_kdebski@on.eu
5	11_akot_akot@kot.pl

Źródło: opracowanie własne

Działanie bardzo podobne do funkcji CAST() ma w swojej najprostszej postaci funkcja CONVERT(). Funkcje te różnią się składnią. Funkcja CONVERT() przyjmuje dwa parametry: typ wyjściowy oraz wartość do konwersji. Przykład ilustruje listing 8.8.

Listing 8.8. Przykłady wykorzystania funkcji `CONVERT()`

```
SELECT CONVERT(int, '42');
SELECT CONVERT(decimal(9,2), '42.42');
```

Źródło: opracowanie własne

Funkcja CONVERT() przyjmuje również opcjonalny trzeci argument określający format. Jego format jest predefiniowany i określa datę oraz czas. Listę przykładowych formatów przedstawia tabela 8.7.

Tabela 8.7. Wybrane formaty daty i czasu dla funkcji `CONVERT()`

Numer formatu	Liczba cyfr – rok	Format godziny	Opis	Przykład
0	2	12	Format domyślny	Aug 16 2012 1:05PM
2	2		ANSI	12.08.16
6	2		Tylko data	16 Aug 12
8		24	Tylko czas	13:05:35
9	4	12	Format domyślny, milisekundy	Aug 16 2012 1:05:35:123PM
11	2		Japonia	12/08/16
13	4	24	Europa	16 Aug 2012 13:05:35:123
105	4		Włochy	16-08-2012

Źródło: (Turley, Wood, 2009)

Funkcja CONVERT() jest więc polecana do konwersji typów daty oraz czasu. Przykładowe jej zastosowania przedstawia listing 8.9. Wynik zapytania z listingu prezentuje tabela 8.8.

Listing 8.9. Przykład wykorzystania funkcji `CONVERT()` do konwersji daty

```
SELECT 'Default Date: '
+ CONVERT(varchar(50), GETDATE(), 100) AS date;
```

Źródło: opracowanie własne

Tabela 8.8. Wynik zapytania z listingu 8.9.

	date
1	Default Date: Aug 16 2012 3:08PM

Źródło: opracowanie własne

Kolejną wartą uwagi funkcją jest STR(). Jest ona wykorzystywana do konwersji wartości numerycznych na format string. Funkcja przyjmuje trzy argumenty: wartość numeryczną, całkowitą długość oraz liczbę cyfr po przecinku, do której ma być zaokrąglony wynik. Przykład wykorzystania funkcji STR() (Turley, Wood, 2009) przedstawia listing 8.10., a wynik tej operacji przedstawia tabela 8.9.

Listing 8.10. Przykład wykorzystania funkcji `STR`

```
SELECT STR(123.456789, 8, 4) AS wynik
```

Źródło: opracowanie własne

Tabela 8.9. Wynik zapytania z listingu 8.10.

	wynik
1	123.4568

Źródło: opracowanie własne

8.5. FUNKCJE DATY

Kolejną grupą funkcji są funkcje pozwalające na przetwarzanie danych związanych z datą oraz czasem. Do najpopularniejszych należą (Turley, Wood, 2009):

- `DATEADD()` umożliwiająca dodanie lub odjęcie określonego interwału czasowego do wartości daty lub czasu. Argumentami funkcji są jednostka czasu, liczba jednostek oraz data.
- `DATEDIFF()` określająca wyrażony w zadanych jednostkach przedział czasowy pomiędzy dwoma datami. Argumentami funkcji są: jednostka czasu oraz daty wyrażone w tych samych formatach.
- `DATEPART()` zwracająca wartość liczbową określającą wybraną część daty (np. numer miesiąca). Bliźniaczą funkcją jest `DATENAME()`, która zwraca pełną, słowną nazwę określonej części daty (np. nazwę miesiąca). Obie funkcje przyjmują dwa argumenty: nazwę interwału czasowego oraz datę.
- `GETDATE()` oraz `GETUTCDATE()` zwracające aktualną datę oraz czas. Funkcja `GETUTCDATE()` bierze pod uwagę ekwiwalent czasu *Universal Time Coordinate* zgodnego z ustawieniami serwera.
- `DAY()`, `MONTH()` oraz `YEAR()` pozwalające na wyświetlenie wartości liczbowej odpowiadającej określonej części daty. Korzystając z kombinacji tych funkcji można w dowolny sposób sformatować wyświetlaną datę. Dostępne interwały czasowe przedstawione zostały w tabeli 8.10.

Tabela 8.10. Interwały czasowe

Nazwa	**Wartości**
Year	Year, yyyy, yy
Quarter	Quarter, qq, q
Month	Month, mm, m
Day of the year	DayOfYear, dy, y
Day	Day, dd, d
Week	Week, wk, ww
Hour	Hour, hh
Minute	Minute, mi, n
Second	Second, ss, s
Milisecond	Millisecond, ms

Źródło: (Turley, Wood, 2009)

Przykłady zastosowania funkcji DATEADD(), DATEPART() oraz DATENAME() przedstawiają odpowiednio listingi 8.11., 8.12. i 8.13. Odpowiednio w tabelach 8.11. – 8.13. umieszczone zostały wyniki tych zapytań.

Listing 8.11. Przykład wykorzystania funkcji `DATEADD`

```
SELECT DATEADD(Day, 90, '10-07-2012') AS date
```

Źródło: opracowanie własne

Tabela 8.11. Wynik zapytania z listingu 8.11.

	Date
1	2013-01-05 00:00:00.000

Źródło: opracowanie własne

Listing 8.12. Przykład wykorzystania funkcji `DATEPART`

```
SELECT DATEPART(month, '07-14-2012') AS date
```

Źródło: opracowanie własne

Tabela 8.12. Wynik zapytania z listingu 8.12.

	Date
1	7

Źródło: opracowanie własne

Listing 8.13. Przykład wykorzystania funkcji `DATENAME`

```
SELECT DATENAME(month, '07-14-2012') AS date
```

Źródło: opracowanie własne

Tabela 8.13. Wynik zapytania z listingu 8.13.

	Date
1	July

Źródło: opracowanie własne

Przykład zapytania zwracającego niestandardowo sformatowaną datę zawiera listing 8.14 i tabela 8.14. Wykorzystana została konkatenacja wyników różnych funkcji konwersji.

Tabela 8.14. Wynik zapytania z listingu 8.14.

	Date
1	Rok: 2012, Miesiac: 8, Dzien: 16

Źródło: opracowanie własne

Listing 8.14. Przykład niestandardowego sformatowania daty

```
SELECT 'Rok: ' + CONVERT(varchar(4), YEAR(GETDATE()))
+ ', Miesiąc: ' + CONVERT(varchar(2), MONTH(GETDATE()))
+ ', Dzień: ' + CONVERT(varchar(2), DAY(GETDATE()))
AS date
```

Źródło: opracowanie własne

8.6. FUNKCJE MANIPULACJI CIĄGÓW ZNAKÓW

Manipulację ciągami znaków można przeprowadzać z wykorzystaniem funkcji: ASCII(), CHAR(), UNICODE() oraz NCHAR(). Funkcje ASCII() oraz CHAR() oparte są o standard ASCII. Pierwsza z nich zwraca numer zadanego znaku w tablicy ASCII, druga przeprowadza operację odwrotną. Funkcja UNICODE() jest odpowiednikiem funkcji CHAR() dla standardu Unicode. Warto zauważyć, że dla MS SQL Serwera typy nchar oraz nvarchar, a także ntext i nvarchar (MAX) wspierają znaki Unicode, więc bez przeszkód współpracować mogą z funkcją CHAR().

Istnieje także kilka innych funkcji przydatnych w zadaniach przetwarzania tekstu. Należą do nich (MSDN, String Functions):

- `LEN()` jest funkcją określającą długość ciągu znakowego (typu string) zadanego w parametrze.
- `CHARINDEX()` zwracającą miejsce występowania zadanego ciągu znaków w ciągu wzorcowym. Funkcja ta przyjmuje dwa argumenty: szukany ciąg znaków oraz ciąg wzorcowy.
- `LEFT()` oraz `RIGHT()` to bliźniacze funkcje zwracające podciąg znaków o określonej długości. Funkcja `LEFT()` zwraca podciąg rozpoczynający się z lewej strony zadanego ciągu znaków, funkcja `RIGHT()` – z prawej strony. Z kolei funkcja SUBSTRING() zwraca podciąg znaków określony przez indeks początkowy oraz długość.
- Funkcje `LOWER()` oraz `UPPER()` konwertują zadany w parametrze ciąg znaków, zmieniając go odpowiednio na małe oraz duże litery.
- `LTRIM()` oraz `RTRIM()` usuwają z zadanego ciągu znaków białe znaki – odpowiednio z lewej i z prawej strony aż do odnalezienia znaków znaczących.
- `REPLACE()` jest funkcją zastępującą określony znak ciągu znaków innym znakiem lub ciągiem znaków. Podobne działanie ma funkcja `STUFF()`, która umożliwia zmianę określonego podciągu znaków innym, niezależnie od długości wstawianego ciągu znaków. Z kolei funkcje `REPLICATE()` oraz `SPACE()` pozwalają wypełnić ciągi znaków określoną liczbą powtórzeń wskazanego znaku (funkcja `REPLICATE()`) lub spacji (funkcja `SPACE()`).

Listingi 8.15. – 8.17. przedstawiają przykłady wykorzystania kilku wymienionych funkcji – odpowiednio CHARINDEX, LEFT, STUFF.

Listing 8.15. Przykład wykorzystania funkcji `CHARINDEX`

```
SELECT CHARINDEX('to', 'testowy napis') AS napis
```

Źródło: opracowanie własne

Tabela 8.15. Wynik zapytania z listingu 8.15.

	napis
1	4

Źródło: opracowanie własne

Listing 8.16. Przykład wykorzystania funkcji `LEFT`

```
SELECT LEFT('testowy napis', 4)
```

Źródło: opracowanie własne

Tabela 8.16. Wynik zapytania z listingu 8.16.

	napis
1	test

Źródło: opracowanie własne

Listing 8.17. Przykład wykorzystania funkcji `STUFF`

```
SELECT STUFF('Zamówienie obejmuje x sztuk', 21, 1, '8')
+ ' '
+ STUFF('o łączniej kwocie xx.xx zł. ', 19, 5, '115.90')
```

Źródło: opracowanie własne

Tabela 8.17. Wynik zapytania z listingu 8.17.

	napis
1	Zamówienie obejmuje 8 sztuk o laczniej kwocie 115.90 zł.

Źródło: opracowanie własne

8.7. FUNKCJE MATEMATYCZNE I FUNKCJE METADANYCH

Przydatnym zbiorem funkcji są również te pozwalające na przeprowadzanie operacji matematycznych. Najbardziej używane funkcje z tej grupy zostały zestawione w tabeli 8.18.

Tabela 8.18. Funkcje matematyczne

Nazwa	Opis
ABS()	Wartość bezwzględna
CEILING()	Zaokrąglenie w górę
FLOOR()	Zaokrąglenie w dół
ROUD()	Zaokrąglenie do określonej precyzji
EXP()	Logarytm naturalny podniesiony do określonej potęgi
LOG()	Logarytm o podstawie 2
LOG10()	Logarytm o podstawie 10
POWER()	Podniesienie do potęgi
SQUARE()	Podniesienie do potęgi 2
SQRT()	Pierwiastek kwadratowy
PI()	Wartość liczby PI jako typ float
RAND()	Liczba określona w oparciu o algorytm losujący
SIGN()	Znak liczby (-1 lub 1)

Źródło: (Turley, Wood, 2009)

Wykorzystanie tych funkcji nie jest skomplikowane. Listing 8.18. przedstawia użycie przykładowej funkcji FLOOR, która odpowiedzialna jest za zaokrąglanie podanej w parametrze liczby w dół. Wynik umieszczono w tabeli 8.19.

Listing 8.18. Przykład wykorzystania funkcji `FLOOR`

```
SELECT FLOOR(42.78) AS wynik
```

Źródło: opracowanie własne

Tabela 8.19. Wynik zapytania z listingu 8.18.

	wynik
1	42

Źródło: opracowanie własne

Ostatnią z omawianych w tym rozdziale grupą funkcji są tzw. funkcje metadanych przechowujące informacje o konfiguracji serwera oraz o ustawieniach bazy danych, w tym o stanie poszczególnych obiektów bazy danych. Zestawienie najpopularniejszych i najbardziej przydatnych funkcji zamieszczono w tabeli 8.20.

Tabela 8.20. Funkcje metadanych

Nazwa	Opis
COL_LENGTH()	Długość określonej w parametrze kolumny
COL_NAME()	Nazwa kolumny o id zadanym w parametrze
COLUMNPROPERTY()	Flaga określająca stan takich właściwości kolumny jak: AllowsNull, IsComputed, Precision, Scale IsCursorType, IsIdentity, IsIndexable, itp.

DATABASEPROPERTY()	Flaga określająca stan takich właściwości bazy danych jak: IsAnsiNullDefault, IsAnsiNullsEnabled, IsAutoClose, IsAutoCreateStatistics, IsAutoShrink, IsDboOnly, IsDetached, IsEmergencyMode, IsFulltextEnabled, IsInRecovery, IsInStandBy, IsNotRecovered, IsOffline, IsReadOnly, IsShutDown, Version, etc.
DB_ID()	Określa id bazy danych na podstawie nazwy.
DB_NAME()	Zwraca nazwę bazy danych na podstawie id.
INDEXKEY_PROPERTY()	Zwraca flagę wskazującą na właściwości klucza indeksującego (ang. *index key*) takie jak ColumnId oraz IsDescending.
INDEXPROPERTY()	Zwraca flagę wskazującą na właściwości indeksu takie jak: IndexDepth, IndexFillFactor, IndexID, IsAutoStatistics, IsClustered, IsFulltextKey, IsPageLockDisallowed, IsRowLockDisallowed, IsStatistics, IsUnique, etc.
OBJECT_ID()	Określa id obiektu bazy danych na podstawie nazwy
OBJECT_NAME()	Zwraca nazwę obiektu bazy danych na podstawie id.
OBJECTPROPERTY()	Zwraca informację dotyczącą różnych właściwości obiektów bazy danych. Dostępne właściwości różnią się w zależności od typu obiektu. Przykładowo mogą to być: CnstIsColumn, CnstIsDeleteCascade, CnstIsDisabled, CnstIsUpdateCascade, ExecIsAfterTrigger, ExecIsDeleteTrigger, ExecIsFirstDeleteTrigger, ExecIsFirstInsertTrigger, ExecIsFirstUpdateTrigger, ExecIsInsertTrigger, ExecIsStartup, ExecIsTriggerDisabled, HasInsertTrigger, HasUpdateTrigger, IsCheckCnst, IsConstraint, IsDefault, IsExecuted, IsForeignKey, IsIndexable, IsIndexed, IsPrimaryKey, IsProcedure, IsRule, IsSystemTable, IsTable, IsTrigger, IsUserTable, IsView, OwnerId, TableDeleteTrigger, TableDeleteTriggerCount, TableFulltextKeyColumn, TableFullTextPopulateStatus, TableHasCheckCnst, TableHasActiveFulltextIndex, TableHasDefaultCnst, TableHasDeleteTrigger, TableHasForeignKey, TableHasForeignRef, TableHasIdentity, TableHasIndex, TableHasInsertTrigger, TableHasPrimaryKey, TableHasTextImage, TableHasTimestamp, TableHasUpdateTrigger, TableInsertTrigger, TableInsertTriggerCount, TableIsFake, TableIsPinned, TableTextInRowLimit, etc.

Źródło: (Turley, Wood, 2009; Microsoft MSDN)

Listing 8.19. i tabela 8.21. przedstawiają odpowiednio zapytanie z wykorzystaniem funkcji meta danych: COLUMNPROPERTY i OBJECT_ID oraz jego wynik (MSDN, Columnproperty).

Listing 8.18. Przykład wykorzystania funkcji metadanych

```
SELECT COLUMNPROPERTY(OBJECT_ID('Users','table'), 'Id_user', 'IsIdentity') AS 'IsIdentity';
```

Źródło: opracowanie własne

Tabela 8.21. Wynik zapytania z listingu 8.18.

	IsIdentity
1	1

Źródło: opracowanie własne

Metadane dotyczące obiektów bazodanowych mogą być także wydobywane z odpowiednich perspektyw systemowych.

8.8. PODSUMOWANIE

W rozdziale przedstawiono podział, zastosowanie oraz przykłady wykorzystania różnych rodzajów funkcji oraz zmiennych systemowych. Funkcje systemowe takie jak funkcje konwersji, daty czy funkcje matematyczne są przydatne w tworzeniu zaawansowanych i średniozaawansowanych zapytań. Funkcje metadanych a także zmienne systemowe mają zastosowanie w zadaniach administracyjnych.

Zarówno funkcji jak i zmiennych jest wiele i ich pełny opis wykracza poza zakres tego skryptu. Szczegółowe opisy oraz pomoc można znaleźć w dokumentacji Microsoft.

8.9. PYTANIA KONTROLNE

1. Wymień i scharakteryzuj grupy funkcji systemowych i zmiennych.
2. Wyjaśnij zastosowanie funkcji agregujących.
3. Wymień przykładowe funkcje formatowania daty.

9

Transact SQL – podstawy

Cel

W rozdziale omówiono podstawy języka Transact SQL i przedstawiono przykłady jego użycia. Przedstawiono zastosowanie zmiennych prostych, użycie zapytania `SELECT`, jak także podstawowych instrukcji warunkowych, wyboru oraz iteracyjnych. Rozdział zawiera wiele przykładów programów napisanych w języku T-SQL ze szczegółowymi opisami, a także z uzyskanymi wynikami.

Plan

1. Zmienne lokalne
2. Polecenie `SELECT`
3. Instrukcja `IF`
4. Instrukcja `CASE`
5. Pętla `WHILE`

9.1. WSTĘP

Transact SQL, w skrócie T-SQL, jest językiem rozszerzającym język SQL. Przy pomocy języka T-SQL można (Kalbarczyk, 2010) (Transact SQL User's Guide, 2005):

- definiować zmienne i przypisywać im wartości,
- tworzyć instrukcje warunkowe,
- tworzyć pętle,
- tworzyć kursory,
- tworzyć funkcje,
- tworzyć procedury wbudowane,
- tworzyć wyzwalacze.

Język T-SQL został wykupiony przez firmę Microsoft i wdrożony w MS SQL Server. Wszystkie aplikacje, które komunikują się z instancją serwera SQL, wysyłają komunikat do serwera z użyciem tego języka, bez względu na interfejs użytkownika (MSDN Library, Transact-SQL Reference).

Język T-SQL ewoluuje, w nowszych wersjach pojawiają się jego zmiany, udogodnienia oraz rozszerzenia.

9.2. ZMIENNE LOKALNE

Jedną z podstawowych możliwości w języku T-SQL jest tworzenie zmiennych i ich późniejsze użycie. Zmienną deklaruje się poleceniem `DECLARE` (Kalbarczyk, 2010) (Transact SQL User's Guide, 2005) (MSDN, Transact-SQL Reference) (MSDN, Declare @local_variable). Nazwa zmiennej jest poprzedzona znakiem @.

Listing 9.1. przedstawia prosty program wyświetlający tekst przypisany do jednej zmiennej. Po nazwie zmiennej należy podać typ danej. W tym przypadku jest to typ tekstowy o maksymalnej długości znaków równej 30. Do przypisania wartości zmiennej można użyć polecenia `SET` (MSDN, SET @local_variable), po którym, po znaku równości, podawana jest wielkość przyporządkowana do nazwy zmiennej. Polecenie `PRINT` (MSDN, PRINT) wypisuje nazwę zmiennej na ekranie. W przedstawionym przykładzie zostanie wyświetlony tekst przypisany do zmiennej (listing 9.1).

Listing 9.1. Program z jedną zmienną typu tekstowego

```
DECLARE @tekst varchar(30)
SET @tekst ='Pierwszy przyklad'
PRINT @tekst
```

Źródło: opracowanie własne

Można także przypisać wartość zmiennej już podczas jej deklaracji (MSDN, Declare @local_variable). Zmodyfikowany przykład z listingu 9.1 pokazujący przypisanie wartości zmiennej podczas jej deklaracji został pokazany na listingu 9.2. Działanie tego programu jest identyczne w porównaniu do wcześniej zaprezentowanego.

Listing 9.2. Program z jedną zmienną typu tekstowego z bezpośrednim przypisaniem wartości

```
DECLARE @tekst varchar(30) ='Pierwszy przyklad'
PRINT @tekst
```

Źródło: opracowanie własne

Przykład przedstawiający użycie dwóch zmiennych został przedstawiony na listingu 9.3. Dwie zmienne (@a oraz @b) przechowują dane typu całkowitego (`int`). Zostały one zadeklarowane i oddzielone przecinkami. Z użyciem polecenia `SET` zostały przypisane im wartości, odpowiednio 1 oraz 8. Następnie poleceniem `PRINT` (MSDN, PRINT) wyświetlono ich sumę. Jako wynik otrzymano: *Suma liczb 1 oraz 8 wynosi 9*.

Listing 9.3. Program wyświetlający sumę dwóch liczb całkowitych

```
DECLARE @a int, @b int
SET @a = 1
SET @b = 8
PRINT 'Suma liczb ' + convert(varchar(2), @a) + ' oraz ' +
      convert(varchar(2), @b) + ' wynosi ' +
      convert(varchar(2), @a + @b)
```

Źródło: opracowanie własne

W poleceniu `PRINT` należało przeprowadzić konwersję danych z typu całkowitego na typ tekstowy, aby możliwe było połączenie łańcuchów. Do tego celu użyto funkcji wbudowanej `CONVERT` (MSDN, Cast and Convert), z dwoma argumentami: typem danych, na który zmienna zostanie zamieniona oraz wartością, którą konwertowano.

Kolejny przykład (listing 9.4) przedstawia przypisanie bieżącej daty do zmiennej `@data` i jej wyświetlenie. Ponieważ funkcja `GETDATA()` (MSDN, Getdate) pobiera zarówno datę, jak i godziny i minuty, zastosowano funkcję `LEFT` (MSDN, Left), która obcina z podanego ciągu określoną liczbę znaków. W ten sposób wyświetlona została jedynie data (dla angielskiego formatu daty może być to przykładowo: *Dzisiaj jest data: Aug 16 2012*).

Listing 9.4. Program wyświetlający bieżącą datę

```
DECLARE @data nvarchar(40);
SET @data = LEFT(GETDATE(), 11);
PRINT 'Dzisiaj jest data: ' + @data;
```

Źródło: opracowanie własne

9.3. POLECENIE SELECT

W programowaniu w języku T-SQL często stosuje się zapytania wybierające SELECT (MSDN, SELECT @local_variable). Zapytania te mogą zostać rozszerzone o zastosowanie zmiennych, np. używanych jako wartości podanych ograniczeń wyszukiwanych danych. Zmienne lokalne muszą zostać zadeklarowane w sekcji DECLARE, a następnie umieszczone w wyrażeniu.

Na listingu 9.5 umieszczono przykład wyświetlający jako tekst pobrane z zapytania nazwisko osoby z tabeli Users użytkownika o identyfikatorze 2. W tym celu najpierw zadeklarowano lokalną zmienną o nazwie @nazwisko. Następnie, w zapytaniu SELECT, do zmiennej przypisano wartość nazwiska zwróconą przez zapytanie przy pomocy znaku równości. Dalszą część zapytania SELECT konstruuje się zgodnie z zasadami języka T-SQL.

Listing 9.5. Program wyświetlający nazwisko użytkownika

```
DECLARE @nazwisko varchar(30)
SELECT @nazwisko = surname
FROM Users
WHERE id_user = 2;
PRINT @nazwisko;
```

Źródło: opracowanie własne

Wykonanie zapytania z listingu 9.5 nie jest jednoznaczne z wyświetleniem pobranych danych. Przykładowo istnieje możliwość wyświetlenia zmiennej poleceniem PRINT. Można także, jak jest to pokazane na listingu 9.6, wyświetlić z użyciem polecenia SELECT. Wtedy dane są wyświetlane analogicznie jak po wykonaniu zapytania.

Listing 9.6 jest rozszerzeniem wcześniejszego przykładu. Zdefiniowane zostały dwie zmienne, odpowiednio nazwisko i imie. Po wykonaniu zapytania wybierającego imię i nazwisko użytkownika o identyfikatorze 2, zostaną one wyświetlone podobnie jak w zapytaniu. Dodatkowo zastosowano aliasy wyświetlanych kolumn.

Listing 9.6. Program wyświetlający nazwisko i imię użytkownika

```
DECLARE @nazwisko varchar(30), @imie varchar(20)
SELECT @nazwisko = surname, @imie = name
FROM Users
WHERE id_user = 2;
SELECT @nazwisko AS Nazwisko, @imie AS Imie;
```

Źródło: opracowanie własne

Przypisanie wyniku zapytania do zmiennej można także wykonać stosując podzapytania (MSDN, SELECT @local_variable). Przekształcenie pierwszego przykładu (listing 9.5) z użyciem podzapytania zostało przedstawione na listingu 9.7.

Listing 9.7. Program wyświetlający nazwisko użytkownika z użyciem podzapytania

```
DECLARE @nazwisko varchar(30)
SELECT @nazwisko = (SELECT surname
FROM Users
WHERE id_user = 2)
SELECT @nazwisko AS Nazwisko;
```

Źródło: opracowanie własne

Podzapytanie jest ujęte w nawiasy okrągłe. Zwracana przez podzapytanie wartość jest przypisywana do zmiennej za pomocą operatora równości. W przypadku, gdy podzapytanie nie zwróci żadnego rekordu, do zmiennej podstawiana jest wartość `NULL`.

Zmienne zastosowane w zapytaniu `SELECT` mogą zostać użyte jako parametry ograniczające wyszukiwane rekordy.
Na listingu 9.8 przedstawiono przykład programu wyświetlającego tytuły i opisy zdjęć, których identyfikatory są mniejsze niż wartość przypisana do zmiennej @id.
Ograniczenie z użyciem tej zmiennej zostało umieszczone w klauzuli `WHERE`.
Zwrócone rekordy posortowano alfabetycznie według nazwy tytułu. Rekordy, które zwrócił program z listingu 9.8 zostały przedstawione w tabeli 9.1.

Listing 9.8. Program wyświetlający tytuły i opisy zdjęć

```
DECLARE @id int = 5
SELECT title, description
FROM Photos
WHERE id_photo < @id
ORDER BY title;
```

Źródło: opracowanie własne

Tabela 9.1. Rekordy zwrócone przez program z listingu 9.8

title	*description*
Konferencja pasjonatów modelarstwa	Zapraszamy do udziału w konferencji, 12 – 14 czerwca 2012
Mój ogród	Wiosną czas na porządki
Pocztówka z Chorwacji	Z serii miasta Europy
Zachód słońca nad Wisłą	Wspomnienie z wakacji

Listing 9.9. Program wyświetlający komentarze osób o nazwisku rozpoczynającym się literą N

```
DECLARE @wzor varchar(4) = 'Now%'
SELECT surname, name, comment, comment_create_date
FROM Comments c Join Users u on u.id_user = c.id_user
WHERE surname LIKE @wzor
ORDER BY surname;
```

Źródło: opracowanie własne

Kolejny przykład z zastosowaniem zmiennej użytej do ograniczania rekordów został przedstawiony na listingu 9.9. Do zmiennej *@wzor* została przypisana wartość `'Now%'`. W zapytaniu wyszukującym imię, nazwisko, komentarz oraz datę jego utworzenia wyszukiwanie ograniczono do tych użytkowników, których nazwiska rozpoczynają się od podanego ciągu (do tego celu użyto operatora `LIKE`). Zapytanie to zwróciło trzy rekordy, które zostały umieszczone w tabeli 9.2. Wszystkie rekordy zostały posortowane rosnąco według nazwiska.

Tabela 9.2. Rekordy zwrócone przez program z listingu 9.9

surname	*name*	*comment*	*comment_create_date*
Nowak	Iwona	Piękne i dzikie tereny, gratuluje odwagi	2012-01-26 13:21:00.000
Nowicka	Anna	Piękny :)	2011-12-12 09:03:00.000
Nowicka	Anna	Jakie to kwiaty?	2012-03-31 11:05:00.000

Źródło: opracowanie własne

9.4. INSTRUKCJA IF

W języku T-SQL można stosować instrukcję warunkową `IF`. Jej schemat został pokazany na listingu 9.10. Po słowie kluczowym `IF` podawany jest warunek lub wyrażenie (Łuszczyk, 2008). Jeśli jest ono prawdziwe, wykonywana jest *instrukcja1*, a w przeciwnym razie wykonywana jest instrukcja zawarta po słowie `ELSE` (*instrukcja2*). W przedstawionym listingu obie instrukcje są pojedynczymi poleceniami. Jeśli konieczne jest wykonanie więcej niż jednej instrukcji, należy je umieścić w bloku `BEGIN … END`. Pomiędzy nimi znajdować się powinien cały kod wykonywalny.

Listing 9.10. Schemat prostej instrukcji warunkowej

```
IF warunek
   instrukcja1
ELSE
   Instrukcja2
```

Przykład prostej instrukcji warunkowej został pokazany na listingu 9.11. Została zadeklarowana zmienna typu całkowitego, której następnie przydzielono wartość 9. Instrukcja warunkowa sprawdza, czy wartość zmiennej jest podzielna przez 2. Do tego celu zastosowany został operator modulo (%). Jeśli wynik dzielenia modulo przez 2 daje w wyniku zero, czyli gdy liczba jest parzysta, zostanie wyświetlony komunikat: *Liczba jest parzysta*. W przeciwnym wypadku, gdy podany warunek nie jest prawdziwy, zostanie wykonana instrukcja po słowie `ELSE`, czyli wyświetli się tekst: *Liczba jest nieparzysta*. Dla podanej w listingu 9.11 wartości zostanie wyświetlony komunikat, że liczba nie jest parzysta.

Listing 9.11. Program sprawdzający czy liczba jest parzysta

```
DECLARE @liczba int
SET @liczba = 9
IF @liczba % 2 = 0
   PRINT 'Liczba jest parzysta'
ELSE
   PRINT 'Liczba jest nieparzysta';
```

Źródło: opracowanie własne

Instrukcję warunkową można rozszerzyć o kolejne warunki, które umieszczane są po słowach ELSE IF i po których trzeba umieścić instrukcje do wykonania. Dodatkowo można na końcu umieścić opcjonalną instrukcję ELSE, po której umieszcza się instrukcje do wykonania w przypadku, gdy, żaden z warunków nie był prawdziwy. Schemat rozszerzonej instrukcji warunkowej został przedstawiony na listingu 9.12.

Listing 9.12. Schemat złożonej instrukcji warunkowej

```
IF warunek1
   instrukcja1
ELSE IF warunek2
   Instrukcja2
ELSE IF warunek3
   Instrukcja3
…
ELSE instrukcja3
```

Źródło: opracowanie własne

Listing 9.13. Program wyświetlający rodzaj kwiatu

```
DECLARE @kwiat varchar(20)
SET @kwiat = 'róża'
IF @kwiat = 'frezja'
   PRINT 'Kwiatem tym jest frezja'
ELSE IF @kwiat = 'róża'
   PRINT 'Kwiatem tym jest różna'
ELSE IF @kwiat = 'tulipan'
   PRINT 'Kwiatem tym jest tulipan'
ELSE
PRINT 'Inny kwiat';
```

Źródło: opracowanie własne

Na listingu 9.13. przedstawiono program z zastosowaniem złożonej instrukcji warunkowej. Zadeklarowano zmienną o nazwie *kwiat*, której przypisano wartość róża. Następnie umieszczono instrukcję warunkową, która sprawdza, jaka nazwa kwiatu jest przypisana do zmiennej. Jeśli wartość zostanie dopasowana, wyświetlony zostaje odpowiedni komunikat. W przeciwnym razie, wykonywana jest instrukcja po słowie ELSE.

Kolejny przykład wykorzystania instrukcji warunkowej został przedstawiony na listingu 9.14. Program odczytuje z bazy danych liczbę zdjęć, którym nie przyporządkowano żadnych komentarzy.

Listing 9.14. Program sprawdzający liczbę zdjęć bez dodanego komentarza

```
DECLARE @zdjecia int
SELECT @zdjecia = ( SELECT COUNT(*)
                    FROM Photos
                    WHERE id_photo NOT IN
                    (SELECT DISTINT id_photo
                     FROM Comments))
IF @zdjecia = 0
   PRINT 'Brak zdjęć, które nie posiadają dodanego
             komentarza'
ELSE IF @zdjecia <5
   PRINT 'Jest mniej niż 5 zdjęć, które nie posiadają
      dodanego komentarza'
ELSE PRINT 'Jest więcej niż 5 zdjęć, które nie posiadają
            dodanego komentarza';
```

Źródło: opracowanie własne

W pierwszej kolejności deklarowana jest zmienna typu całkowitego. Do niej przypisywana jest liczba zdjęć, które nie posiadają dodanego komentarza. Informacje o liczbie zdjęć pobierane są z bazy danych za pomocą zapytania zawierającego podzapytanie. Zliczana jest liczba wystąpień zdjęć, których identyfikatory nie występują w podzapytaniu. Podzapytanie zwraca wszystkie identyfikatory zdjęć, które istnieją w tabeli Comments, a więc tych zdjęć, do których przypisano komentarz. Aby wszystkie identyfikatory były unikalne, zastosowano klauzulę DISTINCT eliminującą powtarzające się elementy. Za pomocą operatora NOT IN zliczona została liczba identyfikatorów, które nie zostały zwrócone przez zapytanie. Następnie przy pomocy instrukcji warunkowej wyświetlono informację o liczbie zdjęć nieposiadających komentarza (0, mniej lub więcej niż 5).

9.5. KONSTRUKCJA CASE

Poleceniem podobnym do instrukcji warunkowej `IF` jest polecenie `CASE`. Dopasowuje ono wartość zmiennej lub wyrażenia do podanych wartości, a następnie wykonuje odpowiednie instrukcje. Może ono także porównywać wyrażenie logiczne i wykonywać instrukcje. Istnieją więc dwa rodzaje instrukcji `CASE` (MSDN Library, Case):

- proste wyrażenie `CASE` (listing 9.15),
- wyszukujące wyrażenie `CASE` (listing 9.16).

Schemat prostego wyrażenia CASE został przedstawiony na listingu 9.15. (MSDN, Case) (Łuszczyk, 2008).

Listing 9.15. Schemat prostej instrukcji CASE

```
CASE wyrażenie_we
   WHEN wyrażenie_when1 THEN instrukcja1
WHEN wyrażenie_when2 THEN instrukcja2

   ELSE  instrukcja_else
END;
```

Po słowie CASE występuje wyrażenie wejściowe, po słowie WHEN natomiast wyrażenie, do którego porównywane jest wyrażenie wejściowe. Jeśli są one zgodne, wykonywana jest instrukcja umieszczona po słowie THEN. Dodatkowo można użyć klauzuli ELSE, po której umieszczana jest instrukcja wykonywana, gdy wejściowe wyrażenie nie zostanie dopasowanego do żadnego innego wyrażenia. Jest to element opcjonalny. Schemat wyszukującego wyrażenia CASE został przedstawiony na listingu 9.16 (MSDN, Case) (Łuszczyk, 2008).

Listing 9.16. Schemat wyszukującej instrukcji CASE

```
CASE
   WHEN wyrażenie_log1 THEN instrukcja1
   WHEN wyrażenie_log2 THEN instrukcja2
   ELSE  instrukcja_else
END;
```

Po słowie kluczowym CASE nie występuje wyrażenie wejściowe. Natomiast po słowie WHEN należy umieścić wyrażenie logiczne (typu boolean). Jeśli wyrażenie to jest poprawne, wykonywana jest instrukcja umieszczona po słowie THEN. Opcjonalnym elementem jest klauzula ELSE. Każda instrukcja CASE zakończona jest słowem END. Przykład instrukcji wybierającej CASE został przedstawiony na listingu 9.17. Jest to program sprawdzający, czy liczba jest podzielna przez 2, 3 lub przez 5.

Listing 9.17. Program sprawdzający podzielność liczb przez 2, 3 oraz 5

```
DECLARE @liczba int
SET @liczba = 7
PRINT CASE
WHEN @liczba % 2 = 0 THEN  'Liczba jest parzysta'
WHEN @liczba % 3 = 0 THEN  'Liczba jest podzielna przez 3'
WHEN @liczba % 5 = 0 THEN  'Liczba jest podzielna przez 5'
ELSE  'Liczba nie jest podzielna przez 2, 3 oraz 5'
END;
```

Źródło: opracowanie własne

Program (listing 9.17) zawiera zadeklarowaną zmienną, której przyporządkowano wartość 7. Polecenie CASE zostało użyte do sprawdzenia, czy dana liczba jest podzielna przez 2, 3 lub przez 5. Jeśli warunek po WHEN jest prawdziwy, wykonywana jest instrukcja zawarta po słowie THEN, a cała instrukcja wyboru jest kończona. Oznacza to, że warunki sprawdzane są do momentu, aż pierwszy z nich zostanie znaleziony (dopasowany). W przypadku, gdy żaden warunek nie zostanie dopasowany (jak w przypadku wartości zmiennej z listingu 9.17), wykonywana jest instrukcja po słowie ELSE.

Instrukcja PRINT została umieszczona przed wyrażeniem CASE, które jedynie podaje sam tekst do wyświetlenia.

Kolejny przykład instrukcji wybierającej CASE został pokazany na listingu 9.18. Instrukcja CASE została użyta do wyświetlenia słownej oceny wybranego zdjęcia. Zapytanie zwraca średnią ocenę zdjęcia (od 0 do 9) o identyfikatorze 1, a następnie, przy pomocy polecenia CASE wypisuje opis słowny:

- ocena zła,
- ocena dobra,
- ocena bardzo dobra.

Do porównania zbioru wartości został użyty operator `IN`.

Listing 9.18. Program wyświetlający średnią ocenę wybranego zdjęcia

```
DECLARE @rank int
SELECT @rank = (select AVG(rate)
                from Rates
                where id_photo =1);
 PRINT CASE
 when @rank IN (0,1,2) THEN  'Ocena zła'
 WHEN @rank IN (3,4,5) THEN  'Ocena dobra'
 WHEN @rank IN (6,7,8) THEN  'Ocena bardzo dobra'
END;
```

Źródło: opracowanie własne

Prostą instrukcję CASE można użyć w zapytaniu SELECT. Na listingu 9.19 przedstawiony został przykład zapytania wyświetlającego komentarze do zdjęć i ich tytułów posortowanych alfabetycznie rosnąco, według tytułów. Dane pobierane są z tabeli *Comments* bazy danych. Wyświetlane są komentarze, które zostały dodane do poszczególnych zdjęć. W rozpatrywanym przykładzie ograniczono identyfikatory zdjęć do 3 wybranych (1, 2 lub 3). Drugą wyświetlaną kolumną jest tytuł zdjęcia. Tytuł został dopasowany do identyfikatora zdjęcia za pomocą instrukcji CASE. Jeśli wyszukany rekord zawiera jeden z identyfikatorów, instrukcja CASE dopasowuje tytuły zdjęć do ich identyfikatorów. Przykładowo, jeśli pierwszy zwrócony komentarz należał do zdjęcia o identyfikatorze 3, w kolumnie *Tytuł* wyświetlono tekst *Mój ogród*. Wyniki zapytania z listingu 9.19 zostały przedstawione w tabeli 9.3.

Listing 9.19. Program wyświetlający tytuły zdjęć i ich komentarze

```
SELECT Comment AS Komentarz,
     CASE id_photo
        WHEN 1 THEN 'Zachód słońca nad Wisłą'
        WHEN 2 THEN 'Pocztówka z Chorwacji'
        WHEN 3 THEN 'Mój ogród'
    END AS Tytuł
FROM Comments
WHERE id_photo IN (1,2,3)
Order by Tytuł;
```

Źródło: opracowanie własne

Tabela 9.3. Wyniki zapytania z instrukcją CASE z listingu 9.19

Komentarz	***Tytuł***
Jakie to kwiaty?	Mój ogród
Niecierpki – są bardzo odporne i długo kwitną, polecam	Mój ogród
Uwielbiam Chorwację	Pocztówka z Chorwacji
Piękny :)	Zachód słońca nad Wisłą
Warto podróżować po Polsce	Zachód słońca nad Wisłą
Zapraszam do obejrzenia mojej galerii zdjęć	Zachód słońca nad Wisłą

Źródło: opracowanie własne

Instrukcję `CASE` można stosować także w różnych klauzulach zapytania `SELECT`. Przykład przedstawiony na listingu 9.20. pokazuje użycie jej w klauzuli `ORDER BY` w celu posortowania danych. Zapytanie wyświetla rekordy zawierające identyfikator właściciela zdjęcia, jego tytuł i opis użytkowników o identyfikatorach 1 oraz 13. Polecenie `CASE` posłużyło do posortowania danych pierwszego użytkownika według tytułu alfabetycznie, od Z do A. Natomiast dla drugiego użytkownika posortowano dane według tytułu alfabetycznie od A do Z. Wynik zapytania z listingu 9.20 został przedstawiony w tabeli 9.4.

Listing 9.20. Program wyświetlający tytuły zdjęć i ich komentarze

```
SELECT id_user, title, description
FROM Photos
WHERE id_user IN (1,13)
ORDER BY
CASE id_user WHEN  1 THEN title end desc,
CASE id_user WHEN  13 THEN title end asc
```

Źródło: opracowanie własne

Tabela 9.4. Wyniki zapytania z instrukcją `CASE` w klauzuli `ORDER BY`

id_user	*title*	*description*
1	Zachód słońca nad Wisłą	Wspomnienie z wakacji
1	W ciemności – nowy firm A.Holland	Ten film zdobył nominację do Oscara
13	Nowa kolekcja firmy Kazar	Polecam
13	Wiosenna kolekcja firmy H&M	Warto się wybrać na zakupy :)

Źródło: opracowanie własne

9.6. PĘTLA WHILE

W języku T-SQL można także używać instrukcji iteracyjnej, pętli `WHILE`. Schemat tej instrukcji został przedstawiony na listingu 9.21.

Listing 9.21. Struktura instrukcji WHILE (MSDN, While)

```
WHILE Wyrażenie_logiczne

 { instrukcja_sql | blok_instrukcji | BREAK | CONTINUE }
```

Główne elementy tej instrukcji to (MSDN, While):

- `Wyrażenie_logiczne` – wyrażenie, którego wynik zwraca jedną z dwóch wartości: prawdę (`TRUE`) lub fałsz (`FALSE`). Jeśli wyrażenie to zawiera instrukcję `SELECT`, musi ona zostać ujęta w nawiasy.
- `instrukcja_sql|blok_instrukcji` – dowolna instrukcja zgodna z językiem SQL lub T-SQL lub kilka instrukcji zawartych w bloku. Jeśli w instrukcji iteracyjnej występuje więcej niż jedna instrukcja, należy umieścić je w bloku pomiędzy słowami kluczowymi `BEGIN` oraz `END`.
- `BREAK` – instrukcja powodująca wyjście z pętli. W przypadku zagnieżdżonych pętli `WHILE`, wyjście następuje z najbardziej wewnętrznej (najbardziej zagnieżdżonej). Po opuszczeniu pętli, wykonywane są instrukcje umieszczone bezpośrednio po niej, czyli po słowie `END` zamykającym instrukcje należące do pętli.
- `CONTINUE` – instrukcja, która przechodzi do kolejnego obiegu pętli `WHILE`. Jednocześnie pomijane są wszystkie instrukcje po tym słowie.

Przykład wyświetlający nieparzyste liczby z zakresu od 1 do 9 został przedstawiony na listingu 9.22. Zadeklarowano zmienną `licz`, której przypisano początkową wartość 1. Instrukcja iteracyjna `WHILE` została użyta do ograniczenia liczb od 1 do 9. Po przekroczeniu tej wartości przez zmienną `licz`, warunek pętli zwróci wartość `FALSE`, a pętla zakończy się.

Listing 9.22. Program wyświetlający nieparzyste liczby od 1 do 9

```
DECLARE @licz int
 SET @licz = 1
 WHILE @licz < 10
   BEGIN
      PRINT @licz
      SET @licz = @licz + 2
   END
```

Źródło: opracowanie własne

Ponieważ pętla zawiera więcej niż jedną instrukcję, został użyty blok `BEGIN END`, który grupuje wszystkie instrukcje wykonywane w obrębie jednej pętli. Wewnątrz niego zostały umieszczone dwie instrukcje: wyświetlającą wartość zmiennej *licz*, a także ustawiającą jej wartość, zwiększając ją o wartość 2. Program wyświetli w kolejnych liniach liczby nieparzyste od 1 do 9. Kolejny przykład (listing 9.23) przedstawia program wyświetlający średnią ocen dla wybranych identyfikatorów zdjęć.

Listing 9.23. Program wyświetlający średnią ocen wybranych zdjęć

```
DECLARE @id int, @licz int
SET @id = 1
SET @maxid = (select MAX(id_photo)
              from Photos
              )/2
WHILE @id < @maxid
BEGIN
   SELECT AVG(rate)AS SredniaOcena
         FROM Rates
         WHERE id_photo = @id;
      SET @id +=1
END
```

Źródło: opracowanie własne

Tabela 9.5. Wyniki zapytania z instrukcją `WHILE` z listingu 9.23

SredniaOcena
4
SredniaOcena
4
SredniaOcena
4
SredniaOcena
2
SredniaOcena
NULL
SredniaOcena
4
SredniaOcena
5
SredniaOcena
4
SredniaOcena
5

Źródło: opracowanie własne

W programie zadeklarowane zostały dwie zmienne. Pierwsza (`@maxid`) została użyta do warunku pętli. Jej wartość została odczytana jako największa wartość identyfikatora z bazy danych podzielona przez 2. Drugiej wartości przydzielono wartość 1. Służy ona jako zmienna do określenia numeru identyfikacyjnego zdjęcia w zapytaniu `SELECT` wewnątrz pętli. Dopóki warunek pętli jest spełniony, wykonywane są instrukcje: wyświetlenie średniej ocen dla zdjęcia o identyfikatorze `@id`,

a następnie zwiększenie jej wartości o 1. Rekordy, które zostały wyświetlone w wyniku wykonania programu z listingu 9.23, zostały przedstawione w tabeli 9.5.

9.7. PYTANIA KONTROLNE

1. W jaki sposób definiuje się zmienne?
2. Jak można przypisać wartość do zmiennej?
3. Omów instrukcję warunkową `IF`.
4. Omów instrukcję `CASE`. Jak można podzielić te instrukcje?
5. Omów strukturę pętli `WHILE`.

10

Transact SQL – funkcje i procedury

Cel

W rozdziale przedstawiono struktury umożliwiające grupowanie poleceń w języku Transact-SQL: funkcje oraz procedury. Omówiono parametry funkcji i procedur. Podano wiele przykładów ilustrujących tworzenie tych struktur, zarówno bez parametrów, jak także z parametrami wejściowymi i wyjściowymi. Opisano także modyfikowanie funkcji i procedur, ich użycie w programach i usuwanie.

Plan

1. Funkcje
2. Procedury

10.1. WSTĘP

W środowisku MS SQL Server istnieje możliwość definiowania własnych funkcji i procedur przy pomocy składni języka Transact SQL. Tworzenie własnych struktur pozwala na łatwiejsze i efektywniejsze późniejsze ich wykorzystywanie w programach.

Funkcja jest strukturą, która wykonuje określone operacje i zwraca obliczoną wartość. Funkcja pozwala na definiowanie lokalnych zmiennych, jak także zastosowanie innych struktur takich jak: instrukcja warunkowa, instrukcja `case` oraz pętle.

Procedura także grupuje instrukcje do wykonania. Mogą one zależeć od podanych parametrów wejściowych. Struktura ta pozwala na zwracanie wartości do programu wykonywalnego, poprzez zastosowanie parametrów wyjściowych.

Funkcje i procedury są wygodnymi strukturami, które po skompilowaniu, są dostępne w schemacie i z których można wielokrotnie korzystać.

10.2. FUNKCJE

Funkcja jest strukturą, która zawiera grupę instrukcji do wykonania, a na ich podstawie zwraca obliczony rezultat. Wykonywane polecenia mogą zależeć od parametrów podawanych przy wywoływaniu funkcji. Struktura ta umożliwia obliczenie i zarazem zwrócenie rezultatu, który może zostać użyty w innych programach, funkcjach czy procedurach. Rezultat funkcji przyjmuje dwojaką postać: pojedynczej wartości (skalar) lub tabeli.

Funkcję tworzy się poleceniem `CREATE FUNCTION` (MSDN CREATE FUNCTION), które składa się z trzech elementów:

- Słowa kluczowego `CREATE FUNCTION`, po którym podaje się nazwę funkcji oraz opcjonalnie jej parametry,
- słowa kluczowego `RETURNS`, po którym podawany jest rezultat funkcji (zwracaną przez funkcję wartość),
- ciała funkcji (pomiędzy słowami kluczowymi `BEGIN` oraz `END`) zawierającego wszystkie instrukcje do wykonania.

Schemat funkcji został przedstawiony na listingu 10.1.

Listing 10.1. Schemat funkcji (MSDN CREATE FUNCTION)

```
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ][ type_schema_name.
] parameter_data_type
    [ = default ] [ READONLY ] }
    [ ,...n ]
  ]
)
RETURNS return_data_type
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    BEGIN
        function_body
        RETURN returned_expression
    END
[ ; ]
```

Schemat funkcji skalarnej został przedstawiony na listingu 10.1. Jej główne parametry to (MSDN CREATE FUNCTION):

- `schema_name` – nazwa schematu, do którego należy tworzona funkcja.
- `function_name` **–** nazwa funkcji.
- `@parameter_name` – nazwa parametru funkcji. Funkcja może zawierać maksymalnie 2100 parametrów. Podczas wywołania funkcji każdemu parametrowi należy przypisać wartość, chyba, że została zdefiniowana dla niego wartość domyślna. Parametry są zmiennymi lokalnymi, co oznacza, że należą one i posiadają wartość jedynie w obrębie funkcji, w której zostały zdefiniowane. Parametry mogą zostać użyte jako stałe, nie mogą natomiast przyjmować wartości jako nazwy tabel, kolumn lub innych obiektów bazy danych.
- `[ type_schema_name. ] parameter_data_type` – określa typ danych parametru. Dodatkowo, opcjonalnie, można podać nazwę schematu, do którego należy użyty typ. Dla funkcji T-SQL dozwolone są wszystkie typy danych, także te definiowane przez użytkowników, za wyjątkiem typu `timestamp`. Typy nieskalarne (takie jak kursor czy tabela) nie mogą być stosowane jako parametr funkcji. Jeśli nazwa schematu została pominięta, typ danych parametru jest wyszukiwany według: schematu zawierającego nazwy typów danych SQL Server,

domyślnego schematu bieżącego użytkownika w aktualnej bazie danych oraz schematu `dbo` w aktualnej bazie danych.

- `[ =default ]` – wartość domyślna dla parametru funkcji. Jeśli zostanie zdefiniowana wartość domyślna dla parametru, można go pominąć podczas wywoływania funkcji.
- `READONLY` – atrybut tylko do odczytu. Przypisanie tego atrybutu do parametru oznacza brak możliwości jego edycji wewnątrz funkcji. Jeśli typ parametru jest zdefiniowany przez użytkownika typem tabeli, atrybut ten powinien zostać podany.
- `return_data_type` – typ zwracanej przez funkcję wartości skalarnej. Każdy typ jest dozwolony, także zdefiniowany przez użytkownika, za wyjątkiem `timestamp`. Nie można również zwracać wartości typu czy tabela.
- `function_body` – serie poleceń T-SQL, które pozwalają na obliczenie wartości skalarnej.
- `returned_expression` –wartość zwracana przez funkcję: skalarna lub jako wiersze tabeli.

Przykład funkcji skalarnej został przedstawiony na listingu 10.2. Zdefiniowana funkcja, o nazwie `LiczbaOsob()`, oblicza liczbę osób, które dodały minimalnie tyle zdjęć, ile wynosi wartość parametru (`@liczba_zdjec`).

Po zadeklarowaniu typu zwracanej wartości (liczby całkowitej), została zdefiniowana zmienna (`@liczba_os`), do której podstawiono wynik zapytania SQL. Zapytanie to zlicza wystąpienie wszystkich identyfikatorów użytkowników (*id_user*) występujących w tabeli *Photos*. Zastosowane podzapytanie zawęża listę identyfikatorów do tych osób, które dodały dokładnie tyle samo lub więcej zdjęć co wartość parametru funkcji. Ostatnim poleceniem jest zwrócenie wartości przez funkcję (polecenie `RETURN`).

Napisaną funkcję należy skompilować. Jeśli pojawi się komunikat o poprawnym wykonaniu (*Command(s) completed successfully*), funkcja została zapisana i można z niej korzystać, np. podczas prostego polecenia `SELECT`. Przykład programu korzystającego z napisanej funkcji `LiczbaOsób()` został pokazany na listingu 10.3.

Listing 10.2. Funkcja zliczająca liczbę osób, które dodały zdjęcia w zależności od jej parametru

```
CREATE FUNCTION LiczbaOsob(@liczba_zdjec int)
RETURNS int
AS
BEGIN
DECLARE @liczba_os int
 SET @liczba_os= (
 SELECT COUNT(*)
 FROM Photos
 WHERE id_user IN (select id_user
       FROM Photos
       GROUP BY id_user
       HAVING COUNT(*) >= @liczba_zdjec)
 ) RETURN(@liczba_os)
 END;
```

Źródło: opracowanie własne

Listing 10.3. Program używający funkcję LiczbaOsob()

```
DECLARE @i int, @max int, @il int
SET @i = 1
SET @max = 7
WHILE @i <@max
BEGIN
SET @il= (select dbo.LiczbaOsob(@i))
IF( @il != 0)
 Print 'Liczba osób, które dodały ' + convert(varchar(2),@i) +
       ' zdjęć to ' + convert(varchar(3),@il)
ELSE
 Print 'Brak osób, które dodały ' + convert(varchar(2), @i) +
       ' zdjęć'
 SET @i += 1
END;
```

Źródło: opracowanie własne

Program (listing 10.3.) definiuje dwie wartości: pierwszą służącą do sterowania pętlą, a drugą do jej ograniczenia. Wewnątrz instrukcji iteracyjnej do zmiennej przypisano wartość funkcji `LiczbaOsób()` przy pomocy polecenia `SET` i `SELECT`. Jeśli pominięto nazwę schematu podczas definiowania funkcji, należy jej wywołanie poprzedzić nazwą schematu `dbo`. Następnie wyświetlany jest odpowiedni komunikat z użyciem polecenia `PRINT`. Wynik programu został pokazany w tabeli 10.1.

Tabela10.1. Wyniki zapytania programu z listingu 10.3

Liczba osób, które dodały 1 zdjęć to 20
Liczba osób, które dodały 2 zdjęć to 13
Liczba osób, które dodały 3 zdjęć to 7
Liczba osób, które dodały 4 zdjęć to 4
Brak osób, które dodały 5 zdjęć
Brak osób, które dodały 6 zdjęć

Źródło: opracowanie własne

Oprócz wartości skalarnych, funkcja może zwracać także typ tablicowy (ang. *table*). Przykład definicji takiej funkcji został przedstawiony na listingu 10.4. Funkcja `ZdjeciaMiesiaca()` zwraca dane o wszystkich zdjęciach, które zostały wybrane na zdjęcia miesiąca. Dane tych zdjęć zostały pobrane z 2 tabel: *Photos* oraz *Photos_of_month*. Z pierwszej tabeli pobrano identyfikator zdjęcia, tytuł, natomiast z drugiej jego opis. Dodatkowo wyszukane dane powiązano z tabelą użytkowników (*Users*), dzięki czemu możliwe stało się wyświetlenie informacji o nazwisku i imieniu użytkownika, który dane zdjęcie dodał do serwisu. Pobrane dane zostały ograniczone do podanego jako parametr miesiąca (`@NrMiesiaca`).

Listing 10.4. Funkcja zwracająca dane zdjęć miesiąca uzależnioną od parametru – NrMiesiąca

```
CREATE FUNCTION ZdjeciaMiesiaca(@NrMiesiaca int)
RETURNS TABLE
AS
RETURN(
 SELECT p.id_photo, title, pm.description, surname, name
 FROM Photos p JOIN Photos_of_month pm
      ON (p.id_photo = pm.id_photo)
      JOIN Users u ON (u.id_user = p.id_user)
 WHERE Month(date) = @NrMiesiaca
);
```

Źródło: opracowanie własne

Tworzenie funkcji, która zwraca dane pobrane zapytaniem wybierającym, jest podobne do funkcji skalarnej. Najważniejszą różnicą jest wskazanie typu wartości, którą funkcja zwraca, w tym przypadku typu `TABLE`. Następnie w klauzuli `RETURN` należy podać zapytanie, które zostanie zwrócone przez funkcję.

Z utworzonej funkcji można skorzystać przy wyświetleniu danych. Na listingu 10.5 zostało przedstawione użycie funkcji `ZdjeciaMiesiaca()` wywołanej z parametrem miesiąca stycznia. Nazwę funkcji należy poprzedzić nazwą schematu, do której ona należy. Dane otrzymane po wywołaniu funkcji można ograniczać (np. klauzulą `WHERE`) jak także sortować.

Listing 10.5. Przykład użycia funkcji `ZdjeciaMiesiaca()`

```
SELECT * from dbo.ZdjeciaMiesiaca(1)
WHERE surname LIKE 'W%'
ORDER BY title;
```

Źródło: opracowanie własne

Funkcje, które zostały napisane i skompilowane, nie można edytować poleceniem `CREATE FUNCTION`. Do modyfikacji funkcji należy użyć polecenia `ALTER FUNCTION` (MSDN ALTER FUNCTION). Należy wtedy nanieść zmiany do istniejącej funkcji i je zatwierdzić.

Funkcję można także usunąć z bazy danych poleceniem DROP FUNCTION (MSDN, DROP FUNCTION). Przykład usunięcia funkcji `ZdjeciaMiesiaca()` zostało pokazane na listingu 10.6.

Listing 10.6. Usunięcie funkcji `ZdjeciaMiesiaca()`

```
DROP FUNCTION dbo.ZdjeciaMiesiaca;
```

Źródło: opracowanie własne

10.3. PROCEDURY SKŁADOWANE

Procedury składowane (ang. stored procedures) są strukturami, które podobnie jak funkcje, zawierają instrukcje do wykonania (lub zapytanie), a ich definicja przechowywana jest na serwerze. Strukturę tę tworzy się poleceniem CREATE PROCEDURE lub CREATE PROC (MSDN, CREATE PROCEDURE). Procedura może być bezparametrowa lub zawierać podane parametry niezbędne do jej wykonania. Główną różnicą pomiędzy funkcją jest brak zwracania wartości poprzez słowo kluczowe RETURN. Nie oznacza to jednak, że za pomocą procedury nie można zwracać wartości. Istnieje możliwość zwracania wartości, ale przy pomocy parametrów. Schemat polecenia tworzenia procedury został przedstawiony na listingu 10.7.

Listing 10.7. Schemat tworzenia procedury (MSDN, CREATE PROCEDURE)

```
CREATE { PROC | PROCEDURE } [schema_name.] procedure_name
        [ ; number ]
    [ { @parameter [ type_schema_name. ] data_type }
        [ VARYING ] [ = default ] [ OUT | OUTPUT ] [READONLY]
    ] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
[ FOR REPLICATION ]
AS { [ BEGIN ] sql_statement [;] [ ...n ] [ END ] }
[;]
```

Główne parametry polecenia tworzącego procedurę to (MSDN, CREATE PROCEDURE):

- `schema_name` – nazwa schematu, do którego należy tworzona funkcja. Jeśli podany schemat nie istnieje lub nie został podany, procedura zostanie przypisana do domyślnego schematu.
- `procedure_name` – unikalna nazwa procedury. Nie powinno stosować się nazw procedur poprzedzonych przedrostkiem `sp_`, gdyż są one zarezerwowane dla procedur systemowych.
- `number` – opcjonalna liczba całkowita używana do grupowania procedur o tej samej nazwie. Zgrupowane procedury mogą zostać usunięte jednym poleceniem `DROP PROCEDURE`.
- `@parameter_name` – nazwa parametru procedury. Procedura może zawierać maksymalnie 2100 parametrów. Podczas wywołania procedury każdemu parametrowi należy przypisać wartość, chyba, że została zdefiniowana dla niego wartość domyślna. Parametry są zmiennymi lokalnymi, co oznacza, że należą one i posiadają wartość jedynie w obrębie procedury, w której zostały zdefiniowane. Parametry mogą zostać użyte jako stałe, nie mogą natomiast przyjmować wartości jako nazwy tabel, kolumn lub innych obiektów bazy danych. Jeśli w definicji procedury występuje parametr `FOR REPLICATION`, nie można zadeklarować parametrów.
- `[ type_schema_name. ] parameter_data_type` – określa typ danych parametru oraz schemat, do którego typ należy.
- `[ VARYING ]` – określa zestaw zwracanych wartości obsługiwany jako parametr wyjściowy. Parametr jest dynamicznie tworzony przez procedurę. Element ten może zostać zastosowany jedynie do parametrów typu kursora.

- `[ =default ]` – wartość domyślna dla parametru funkcji. Jeśli zostanie zdefiniowana wartość domyślna dla parametru, można go pominąć podczas wywoływania procedury.
- `OUT | OUTPUT` – wskazuje, że parametr jest typu wyjściowego, czyli jego wartość będzie dostępna z miejsca wywołania procedury (zwrócenie wartości). Parametry typu `text`, `ntext` oraz `image` nie mogą zostać jako parametry wyjściowe.
- `READONLY` – atrybut tylko do odczytu. Przypisanie tego atrybutu do parametru oznacza braku możliwości jego edycji wewnątrz funkcji. Jeśli typ parametru jest zdefiniowany przez użytkownika typem tabeli, atrybut ten powinien zostać podany.
- `[ FOR REPLICATION ]` – oznacza, że procedura jest tworzona jako replika (kopia). Procedura tworzona z tym parametrem jest używana jako filtr procedury. W tym ustawieniu nie może posiadać parametrów, a parametr `RECOMPILE` jest ignorowany.
- `{ [ BEGIN ] sql_statement [;] [ ...n ] [ END ] }` – jedno lub wiele wyrażeń SQL składające się na ciało procedury. Polecenia te można umieścić w opcjonalnej strukturze `BEGIN .. END`.

Przykład bezparametrowej procedury został przedstawiony na listingu 10.8. Po nazwie procedury, po słowie `AS`, podawane jest zapytanie SQL. Dodatkowo zostało ono ujęte w blok `BEGIN .. END`. Procedura zawiera definicję zapytania pobierającego dane o zdjęciach, ich użytkowników, posortowanych po nazwie pliku.

Listing 10.8. Utworzenie procedury bezparametrowej Zdjecia

```
CREATE PROCEDURE Zdjecia
AS
BEGIN
        select file_name, description, name, surname
        from Photos p JOIN Users u ON p.id_user = u.id_user
        order by file_name desc
END;
```

Źródło: opracowanie własne

Przykład tworzenia procedury z parametrem został przedstawiony na listingu 10.9. Procedura KomentarzeUzytkownika została zdefiniowana z wymaganym parametrem typu całkowitego, który odnosi się do identyfikatora użytkownika. Procedura pobiera informacje o wszystkich komentarzach wystawionych przez użytkownika o podanym numerze. Parametry procedury mogą zostać podane w nawiasach, ale także poza, jak przedstawiono na listingu 10.9.

Wywołanie procedury KomentarzeUzytkownika zostało pokazane na listingu 10.10. Po słowie kluczowym `EXEC` należy podać nazwę procedury poprzedzoną nazwą schematu, do której ona przynależy. Następnie należy przypisać wartości

wszystkim parametrom procedury. Jeśli parametrów jest więcej niż jeden, należy je oddzielić przecinkami. Przypisanie parametrów może odbywać się poprzez podanie nazwy parametru i przypisanie mu wartości lub podanie samej wartości parametru. W przypadku, gdy stosowane są nazwy parametrów podczas wywoływania procedury, nie trzeba zachowywać kolejności parametrów podanej podczas jej definicji.

Listing 10.9. Utworzenie procedury KomenarzeUzytkownika

```
CREATE PROCEDURE KomenatrzeUzytkownika
                    @id int
AS
BEGIN
        select id_comment, comment, comment_create_date,
               file_name
        from Comments c JOIN Photos p ON c.id_photo=p.id_photo
        where c.id_user = @id
END;
```

Źródło: opracowanie własne

Listing 10.10. Wywołanie procedury KomenarzeUzytkownika

```
EXEC dbo.KomenatrzeUzytkownika @id=1
EXEC dbo.KomenatrzeUzytkownika 1
```

Źródło: opracowanie własne

Utworzenie procedury zawierającej zarówno parametry wejściowe, jak i wyjściowy, przedstawia listing 10.11. Procedura o nazwie KomentarzeOsob zawiera dwa parametry wejściowe oraz jeden parametr wyjściowy. Należy podać imię i nazwisko osoby, aby wyświetlić jej wszystkie komentarze, które umieściła. Dodatkowo procedura zlicza i zwraca liczbę komentarzy podanej osoby. Należy pamiętać o dodaniu słowa `OUT` przy parametrze wyjściowym.

Przykład programu korzystającego z procedury KomentarzeOsob został przedstawiony na listingu 10.12.

Listing 10.11. Tworzenie procedury KomenarzeOsob

```
CREATE PROCEDURE KomentarzeOsob
                        @nazwisko nvarchar(30),
                        @imie nvarchar(20),
                        @liczba_komentarzy int OUT
AS
BEGIN
  select id_comment, comment, comment_create_date, file_name
  from Comments c JOIN Photos p ON c.id_photo=p.id_photo
       JOIN Users u ON u.id_user = c.id_user
  where u.name = @imie AND u.surname = @nazwisko
SET @liczba_komentarzy = (select count(*)
           from Comments c JOIN Users u ON u.id_user=c.id_user
    where u.name = @imie AND u.surname = @nazwisko)
END;
```

Źródło: opracowanie własne

Listing 10.12. Program korzystający z procedury KomentarzeOsob

```
DECLARE @komentarze int
EXEC dbo.KomentarzeOsob 'Nowicka', 'Anna',  @komentarze OUT
IF @komentarze <= 0
   PRINT 'Użytkownik nie dodał jeszcze komentarzy'
ELSE IF @komentarze = 1
   PRINT 'Użytkownik dodał już 1 komentarz'
ELSE
   PRINT 'Użytkownik dodał już ' + CAST(@komentarze AS
         varchar(3)) + ' komentarze'
```

Źródło: opracowanie własne

Program (listing 10.12) deklaruje jedną zmienną typu całkowitego, do której zostanie zwrócona liczba komentarzy użytkownika poprzez procedurę KomentarzeOsob. Należy pamiętać, że wywołanie procedury z parametrem wyjściowym wymaga podania opcji `OUT`.

Następna część programu związana jest z wyświetleniem informacji dla użytkownika o liczbie komentarzy. Została użyta do tego zagnieżdżona instrukcja warunkowa.

Podobnie, jak w przypadku funkcji, napisaną i skompilowaną procedurę można modyfikować poleceniem `ALTER PROCEDURE` (MSDN ALTER PROCEDURE). Skompilowaną procedurę można także usunąć poleceniem `DROP PROCEDURE` (MSDN DROP PROCEDURE).

10.4. PYTANIA KONTROLNE

1. Jakie są różnice pomiędzy funkcją i procedurą?
2. W jaki sposób wywołuje się funkcje i procedury w programie T-SQL?

11

Transact SQL – kursory

Cel

W rozdziale przedstawiono jedną ze struktur języka T-SQL – kursory. Pozwalają one na przetwarzanie kolejnych wierszy danych, zwróconych przez zapytanie. W rozdziale omówiono definiowanie kursora z różnymi parametrami, jego otwarcie, użycie, zamknięcie i zwalnianie.

Rozdział zawiera przykłady ilustrujące definiowanie i zastosowanie kursorów w programach i procedurach.

Plan

1. Deklaracja kursora
2. Otwarcie, zamknięcie i zwalnianie kursora
3. Przetwarzanie kursora

11.1. WSTĘP

Kursor jest strukturą, która umożliwia pobranie listy rekordów zwróconych przez zapytanie, a następnie ich kolejne przetwarzanie, rekord po rekordzie. Pozwala to na indywidualny dostęp do każdego wiersza pobranych danych. Kolejność zwracanych danych jest ściśle związana z utworzonym zapytaniem. Język Transact-SQL umożliwia tworzenie kursorów i korzystanie z nich.

Istnieją cztery typy kursorów, podzielonych według funkcjonalności i korzystania z zasobów systemowych. Są to: `FORWARD_ONLY`, `STATIC`, `DYNAMIC` oraz `KEYSET` (MSDN, DECLARE CURSOR).

W celu użycia kursora, należy: utworzyć zapytanie wybierające, otworzyć kursor, następnie przetworzyć wszystkie jego wiersze, a na końcu go zamknąć. Dodatkowo można w zapytaniu przypisać kursorowi zmienną lub parametr (MSDN, CURSORS).

11.2. DEKLARACJA KURSORA

Deklaracja tworzy definicję kursora. Na listingu 11.1 została przedstawiona definicja kursora. Składa się ona z następujących elementów (MSDN, DECLARE CURSOR):

- `cursor_name` – określa nazwę definiowanego kursora. Musi spełniać warunki nazewnictwa w środowisku MS SQL Server.
- `LOCAL` – określa dostępność kursora. Dostęp lokalny oznacza, że kursor jest dostępny w obrębie, w którym był definiowany, np.: w procedurze czy wyzwalaczu. Kursor może być natomiast przekazany jako parametr wyjściowy procedury (`OUTPUT`). Kursor jest deaktywowany, gdy kończona jest struktura, do której należy (np. procedura czy wyzwalacz), poza wyjątkiem, gdy jest przekazywany jako parametr wyjściowy.
- `GLOBAL` – definiuje globalny dostęp kursora względem połączenia, co oznacza, że zostanie on usunięty (ang. deallocated) podczas kończenia połączenia. W trakcie połączenia można z niego korzystać, np. wewnątrz procedur.
- `FORWARD_ONLY` – parametr oznacza, że kursor może być przetwarzany jedynie od pierwszego do ostatniego rekordu. W tym ustawieniu można stosować jedynie polecenie `FETCH NEXT` do przetwarzania kolejno pobranych danych.
- `SCROLL` – parametr określa, że wszystkie polecenia przetwarzania kursora (`FIRST`, `LAST`, `PRIOR`, `NEXT`, `RELATIVE`, `ABSOLUTE`) są dostępne. Opcja ta nie może zostać podana, jeśli kursor ma podany parametr `FAST_FORWARD`.

- `STATIC` – parametr oznacza, że definiowany jest kursor, który tworzy tymczasową kopię danych przechowywaną w tabeli `tempdb`. Przetwarzanie odbywa się na kopii danych, dlatego w tym ustawieniu kursora nie jest możliwa modyfikacja oryginalnych danych.
- `KEYSET` – określa, że kolejność rekordów kursora jest ustalona.
- `DYNAMIC` – definiuje kursor, który odzwierciedla wszystkie zmiany wykonane na rekordach kursora, na danych oryginalnych. Dane wartości i ich kolejność może ulec zmianie przy każdym pobraniu rekordu kursora.
- `FAST_FORWARD` **–** określa parametry `FORWARD_ONLY`, `READ_ONLY` z włączoną optymalizacją wykonania. Parametr ten nie może być połączony z parametrem `SCROLL` lub `FOR_UPDATE`.
- `READ_ONLY` – opcja ta zabezpiecza przed modyfikacją danych poprzez kursor.
- `SCROLL_LOCKS` – zapewnia zablokowanie modyfikowanych rekordów, co zapewnia, że oryginalne dane zostają zmienione z powodzeniem.
- `OPTIMISTIC` – określa, że zmiana bądź usunięcie danych kursora nie zostanie zapisana w danych oryginalnych, jeśli rekordy tworzące kursor zostały zmodyfikowane od jego otwarcia.
- `TYPE_WARNING` – określa, że jest wysyłany komunikat do klienta, w przypadku, gdy kursor jest domyślnie przekształcany z żądanego typu do innego.
- `select_statement` – zapytanie zwracające dane, o które definiowany jest kursor. Stosowanie słów takich jak: `COMPUTE`, `COMPUTE BY`, `FOR BROWSE`, oraz `INTO` nie jest dopuszczone.

Listing 11.1. Schemat deklaracji kursora (MSDN DECLARE CURSOR)

```
DECLARE cursor_name CURSOR [ LOCAL | GLOBAL ]
     [ FORWARD_ONLY | SCROLL ]
     [ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]
     [ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]
     [ TYPE_WARNING ]
     FOR select_statement
     [ FOR UPDATE [ OF column_name [ ,...n ] ] ]
[;]
```

Przykład definicji kursora *dane_zdjec*, zawierający wszystkie dane tabeli *Photos* posortowane rosnąco według daty utworzenia, został przedstawiony na listingu 11.2.

Listing 11.2. Definicja kursora dane_zdjec

```
DECLARE dane_zdjec CURSOR
     FORWARD_ONLY
     FOR select * from Photos
         Order by photo_create_date
```

Źródło: opracowanie własne

11.3. OTWARCIE, ZAMKNIĘCIE I ZWOLNIENIE KURSORA

Po podaniu definicji kursora, aby z niego skorzystać w pierwszej kolejności należy go otworzyć. Polecenie otwierające kursor zostało przedstawione na listingu 11.3. Jego parametry to (MSDN OPEN):

- `GLOBAL` – określa, że nazwa kursora jest nazwą globalną.
- `cursor_name` – określa nazwę kursora.
- `cursor_variable_name` – nazwa zmiennej kursora.

Listing 11.3. Polecenie otwierające kursor (MSDN OPEN, MSDN DECLARE CURSOR)

```
OPEN { { [ GLOBAL ] cursor_name } | cursor_variable_name }
```

Polecenie otwierające kursor (zdefiniowany na listingu 11.2) zostało przedstawione na listingu 11.4. Po poleceniu OPEN podawana jest nazwa kursora. Od tego momentu można przetwarzać danymi zwróconymi przez zapytanie.

Listing 11.4. Otwarcie kursora dane_zdjec

```
OPEN dane_zdjec
```

Źródło: opracowanie własne

Zamknięcie kursora następuje po wykonaniu polecenia CLOSE (MSDN CLOSE) przedstawionego na listingu 11.5. Polecenie CLOSE może zostać zastosowane tylko do otwartych kursorów. Natomiast nie może ono być użyte dla kursorów, które jedynie zostały zdefiniowane lub już zostały zamknięte. Zamknięcie kursora zwalnia aktualny zestaw wyników i usuwa wszelkie blokady ustawione przez kursor.

Listing 11.5. Zamknięcie kursora dane_zdjec

```
CLOSE dane_zdjec
```

Źródło: opracowanie własne

Zwalnianie kursora polega na usunięciu referencji kursora oraz zwolnieniu danych, które zostały zdefiniowane przez niego. Zwolnienie kursora wykonuje się poleceniem DEALLOCATE (MSDN DEALLOCATE). Na listingu 11.6 został przedstawiony przykład zwolnienia kursora dane_zdjec.

Listing 11.6. Zwolnienie kursora dane_zdjec

```
DEALLOCATE dane_zdjec
```

Źródło: opracowanie własne

11.4. PRZETWARZANIE KURSORA

Przetwarzanie kursora związane jest z dostępem do kolejnych rekordów zwróconych przez zapytanie definiujące kursor. Dostęp do kolejnych wierszy (rekordów) jest możliwy przy pomocy polecenia FETCH, którego definicja została pokazana na listingu 11.7. Najważniejsze elementy tego zapytania to:

- `NEXT` – zwrócenie kolejnego wiersza w stosunku do bieżącego,
- `PRIOR` – zwrócenie poprzedniego wiersza w stosunku do bieżącego,
- `FIRST` – zwrócenie pierwszego wiersza kursora,
- `LAST` – zwrócenie ostatniego wiersza kursora.

Listing 11.7. Definicja polecenia FETCH (MSDN, FETCH)

```
FETCH
          [ [ NEXT | PRIOR | FIRST | LAST
                    | ABSOLUTE { n | @nvar }
                    | RELATIVE { n | @nvar }
               ]
               FROM    ]
{ { [ GLOBAL ] cursor_name } | @cursor_variable_name }
[ INTO @variable_name [ ,...n ] ]
```

Polecenie `FETCH` pobiera kolejne rekordy danych kursora. W celu ograniczenia wykonywanych instrukcji jedynie do listy zwróconych rekordów należy zastosować element iteracyjny języka T-SQL, np. pętli `WHILE`. Bardzo pomocna jest zmienna `@@FETCH_STATUS` (MSDN FETCH_STATUS), która zwraca informację, że został pobrany ostatni rekord otwartego kursora. Zmienna ta może przyjmować 3 wartości:

- -1 – zwrócono ostatni rekord kursora,
- 0 **–** ostatni rekord znajdował się poza zestawem danych,
- -2 – brakuje ostatniego wiersza.

Parametr ten może zostać zastosowany jako warunek pętli przetwarzającej rekordy. W warunku pętli umieszczany jest warunek: `@@FETCH_STATUS = 0`. Pętla przetwarza rekordy, dopóki nie zostanie zwrócony komunikat o ostatnim wierszu danych.

Na listingu 11.8 został przedstawiony program w języku T-SQL korzystający z kursora dane_zdjec. W programie zadeklarowano 4 zmienne: 3 dotyczące danych pobranych z kursora, a ostatnia w celu oznaczenia kolejnych numerów zdjęć. Następnie zdefiniowano kursor dane_zdjec. Kursor został otwarty i pobrany pierwszy jego wiersz danych. Kolejne rekordy kursora zostały przewarzane w pętli WHILE, która kończy się w przypadku braku kolejnych rekordów kursora. Jeśli opis zdjęcia nie został dodany (zmienna @desc przyjmuje wartość NULL), wyświetlane są wszystkie pozostałe pobrane dane oraz informacja o braku opisu zdjęcia. W przeciwnym wypadku wyświetlane są wszystkie dane, zgodnie z instrukcją PRINT. Po wyświetleniu danych pobierany jest następny rekord kursora oraz zwiększana jest wartość zmiennej @il o 1. Po przetworzeniu wszystkich wierszy kursor jest zamykany i zwalniany.

Przykład, pokazany na listingu 11.9, ilustruje tworzenie procedury, w której zadeklarowany jest kursor, a następnie zastosowany jest do wyświetlania wszystkich zdjęć dodanych w określonym roku. Rok jest określany przy pomocy parametru procedury (@rok).

Listing 11.8. Program z użyciem kursora dane_zdjec

```
DECLARE @file varchar(50),
        @title varchar(255),
        @desc varchar(400),
        @il int
SET @il = 1;
DECLARE dane_zdjec CURSOR
     FORWARD_ONLY
     FOR select file_name, title, description
         from Photos
         order by photo_create_date desc;
OPEN dane_zdjec;
FETCH NEXT FROM dane_zdjec
      INTO @file, @title, @desc;

WHILE @@FETCH_STATUS = 0
BEGIN
   IF @desc != null
      PRINT 'Dane zdjecia ' + convert(varchar(3),@il) + ':'+
            @file + '-' +  @title + '-' + @desc +';';
   ELSE
      PRINT 'Dane zdjecia ' + convert(varchar(3),@il) + ':'+
            @file + '-' +  @title + '- brak opisu;';
   SET @il+=1;
   FETCH NEXT FROM dane_zdjec
         INTO @file, @title, @desc;
END
CLOSE dane_zdjec;
DEALLOCATE dane_zdjec;
```

Źródło: opracowanie własne

Listing 11.9. Definicja procedury z kursorem

```
CREATE PROCEDURE ZdjeciaRoku
            @rok INT
AS
DECLARE @file varchar(50),
        @title varchar(255),
        @desc varchar(400),
        @il int
SET @il = 1;
DECLARE dane_zdjec CURSOR
     FORWARD_ONLY
     FOR select file_name, title, description
     from Photos
     where YEAR(photo_create_date) = @rok;
OPEN dane_zdjec;
FETCH NEXT FROM dane_zdjec
INTO @file, @title, @desc;
PRINT 'Dane zdjęć z roku '+ convert(varchar(4),@rok)
WHILE @@FETCH_STATUS = 0
BEGIN
  IF @desc != null
      PRINT 'Dane zdjecia ' + convert(varchar(3),@il) + ':'+
@file + '-' +  @title
      + '-' + @desc +';';
 ELSE
      PRINT 'Dane zdjecia ' + convert(varchar(3),@il) + ':'+
@file + '-' +  @title
      + '- brak opisu;';
 SET @il+=1;
 FETCH NEXT FROM dane_zdjec
              INTO @file, @title, @desc;
END
```

```
PRINT 'W roku ' + convert(varchar(4),@rok) +' dodano '+
convert(varchar(4),@il-1) + ' zdjec'
CLOSE dane_zdjec;
DEALLOCATE dane_zdjec;
```

Źródło: opracowanie własne

Wywołanie procedury ZdjeciaRoku przedstawia listing 11.9. Utworzona procedura może zostać wielokrotnie użyta w wielu programach, funkcjach i procedurach.

Listing 11.9. Definicja procedury z kursorem

```
DECLARE @rok int
SET @rok = 2012
EXEC ZdjeciaRoku @rok=@rok
```

Źródło: opracowanie własne

11.5. PYTANIA KONTROLNE

1. Wyjaśnij pojęcie kursora.
2. Podaj przykład deklaracji kursora.
3. W jaki sposób korzysta się z kursora – omów otwarcie, zamknięcie i zwalnianie kursora.
4. *Omów przetwarzanie kursora.*

12

Analiza wydajności języka T-SQL

Cel

W rozdziale przedstawiono analizę programów napisanych w języku T-SQL. Analiza dotyczyła zajętości procesora oraz czasu wykonania programów. Analizowano:

- wydajność zapytań skorelowanych,
- wydajność programów prostych oraz zawierających procedurę,
- wydajność zapytań prostych oraz zawierających podzapytania,
- wydajność wybranych poleceń języka T-SQL (`MIN`, `MAX`, `TOP`),
- wydajność kursorów oraz procedur korzystających z kursorów.

Każda grupa analizy została poparta uzyskanymi wynikami i wykresami.

Plan

1. Wydajność zapytań skorelowanych
2. Porównanie wydajności działania procedury i zapytania
3. Wydajność zapytań i podzapytań
4. Wydajność wybranych poleceń
5. Wydajność kursorów

12.1. WSTĘP

Dane otrzymane poprzez wykonanie zapytań, funkcji, procedur i kursorów zależą od ich struktury. Te same zbiory wartości można otrzymać przy pomocy różnych zapytań czy struktur. Dlatego warto sprawdzać wydajność napisach zapytań i programów w celu ich dalszej optymalizacji. Analiza wydajności może dotyczyć aspektów takich jak: czas wykonania zapytania, czas zużycia procesora, liczby operacji zapisu i odczytu z bazy danych.

Analizę wydajności zastosowanych obiektów bazodanowych można przeprowadzić w programie, w języku T-SQL. Aby odczytane dane o wydajności programu były wiarygodne, należy wyczyścić pamięć podręczną (cache) serwera bazodanowego (MSDN DROPCLEANBUFFERS, MSDN FREEPROCCACHE). Polecenia służące do tego celu przedstawiono na listingu 12.1.

Listing12.1. Polecenia czyszczące cache serwera bazodanowego

```
DBCC DROPCLEANBUFFERS WITH NO_INFOMSGS
DBCC FREEPROCCACHE WITH NO_INFOMSGS
```

Pomiar czasu oraz wydajności procesora można zmierzyć w programie poprzez uruchomienie i zatrzymanie poleceń SET STATISTICS TIME (MSDN SET STATISTICS TIME), co ilustruje listing 12.2.

Listing 12.2. Polecenia uruchomienia i zatrzymania poleceń SET STATISTIC TIME

```
Set statistics time ON
Set statistics time OFF
```

12.2. WYDAJNOŚĆ ZAPYTAŃ SKORELOWANYCH

Zapytania skorelowane należą do grupy zapytań złożonych, wymagających dłuższego czasu wykonania oraz większego zużycia zasobów procesora niż zapytania proste. Zapytania skorelowane, w przeciwieństwie do innych zapytań złożonych, wymagają wielokrotnego wywoływania podzapytania (w zależności od korelacji zdefiniowanej w zapytaniu głównym).

Przykład zapytania skorelowanego przedstawia Listing 12.3. Wynik zapytania zwraca listę użytkowników, którzy dodali największą liczbę zdjęć do poszczególnych kategorii.

Średni czas wykonywania podzapytania to 3,1 ms, a średnie obciążenie procesora wyniosło 3%. Wartości odchyleń standardowych dla obu parametrów wynoszą odpowiednio 4,83 i 3,54. Dokładne czasy oraz obciążenia przedstawia tabela 12.1 oraz rysunek 12.1.

Listing 12.3. Zapytanie skorelowane

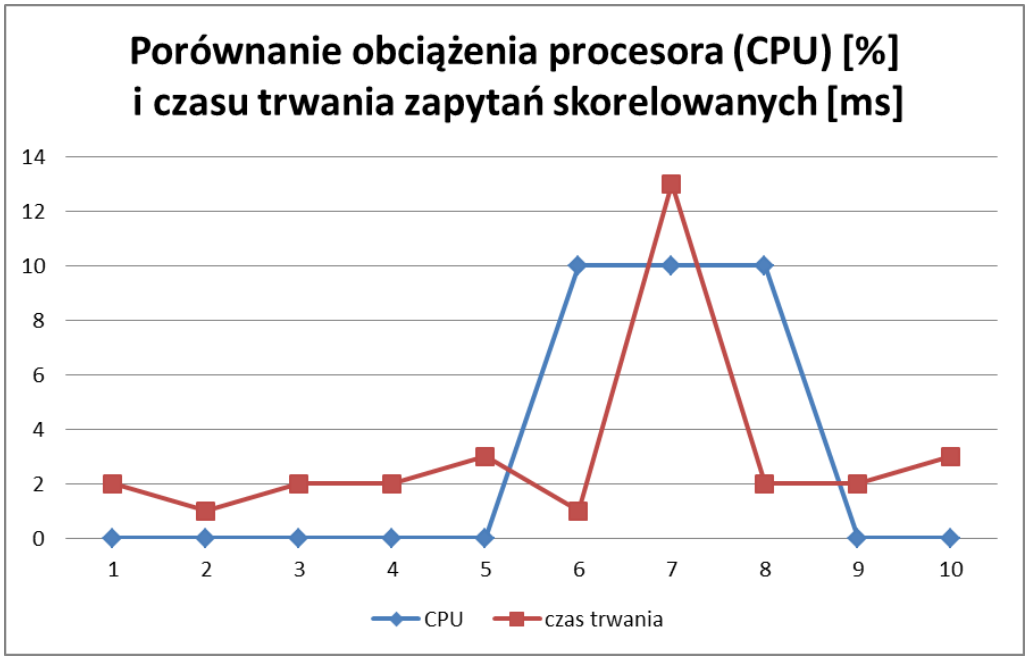
```
SELECT  u.id_user, login, name, surname, k.keyword, COUNT(*)
FROM Users u INNER JOIN Photos p ON u.id_user=p.id_user
INNER JOIN Keyword_to_photos kp ON p.id_photo=kp.id_photo
INNER JOIN Keywords k ON kp.id_keyword=k.id_keyword
WHERE keyword IN ('podróże','moda')
GROUP BY u.id_user, login, name, surname, k.keyword,
k.id_keyword
HAVING COUNT(*)=(SELECT TOP 1 COUNT(*)
FROM Photos p1 INNER JOIN Keyword_to_photos kp1 ON
p1.id_photo=kp1.id_photo
WHERE kp1.id_keyword=k.id_keyword
GROUP BY id_user
ORDER BY COUNT(*) DESC )
```

Źródło: opracowanie własne

Tabela 12.1. Wyniki pomiarów

Lp.	Obciążenie procesora [%]	Czas trwania zapytania [ms]
1	0	2
2	0	1
3	0	2
4	0	2
5	0	3
6	10	1
7	10	13
8	10	2
9	0	2
10	0	3
Wartość średnia	**3**	**3,1**
Odchylenie standardowe	**4,83**	**3,54**

Źródło: opracowanie własne



Rys. 12.1. Porównanie obciążenia procesora (CPU) i czasu trwania zapytań skorelowanych
Źródło: opracowanie własne

12.3. PORÓWNANIE WYDAJNOŚCI DZIAŁANIA PROCEDURY I ZAPYTANIA

Procedura pozwala na grupowanie poleceń języka T-SQL w jedną strukturę, która może być wykorzystywana wielokrotnie przez użytkownika. Istnieje możliwość analizy użycia zasobów procesora oraz czasu niezbędnego na wykonanie programu bez procedury oraz analogicznego programu wywołującego procedurę. Na listingach 12.4 oraz 12.5 zostały przedstawione programy wyświetlające nazwę zdjęć, ich komentarze i dane użytkownika wszystkich fotografii dodanych w wybranym miesiącu (w tym przypadku w styczniu). Listing 12.5 przedstawia zapytanie ujęte w procedurę OcenyZdjec, która posiada jeden parametr (numer miesiąca).

Listing 12.4. Program wyświetlający dane zdjęć, ich użytkowników i ocen w wybranym miesiącu

```
DECLARE
@mies int
BEGIN
SET @mies = 1
select file_name, rate, r.id_photo, r.id_user,login,
rate_creation_date
from Rates r JOIN Users u on r.id_user = u.id_user
JOIN Photos p on r.id_photo = p.id_photo
where MONTH(rate_creation_date) = @mies
order by r.id_photo, rate
END
```

Źródło: opracowanie własne

Listing 12.5. Procedura wyświetlająca dane zdjęć, ich użytkowników i ocen w wybranym miesiącu

```
create procedure OcenyZdjec
       @miesiac int
AS
BEGIN
select file_name, rate, r.id_photo, r.id_user,login,
rate_creation_date
from Rates r JOIN Users u on r.id_user = u.id_user
JOIN Photos p on r.id_photo = p.id_photo
where MONTH(rate_creation_date) = @miesiac
order by r.id_photo, rate
END
```

Źródło: opracowanie własne

Dane uzyskane podczas dziesięciu uruchomień programu bez oraz z procedurą zostały przedstawione w tabelach 12.2 oraz 12.3. Wykresy przedstawiające dane obciążenia procesora oraz czas trwania programów zostały pokazane na rysunkach 12.2 oraz 12.3. W tabelach 12.2 oraz 12.3 umieszczono otrzymane wyniki. Średnie obciążenie procesora wyniosło 2% dla programu bez procedury i 8,1% z procedurą. Odchylenia standardowe są porównywalne. Średni czas wykonywania programu bez procedury to 1,2 ms, a z procedurą 7,8 ms. Odchylenie standardowe jest większe dla programu z procedurą i wynosi w przybliżeniu 1,3, podczas gdy dla pierwszego programu wynosi ono w przybliżeniu 0,4.

Tabela 12.2. Wyniki pomiarów dla programu z listingu 12.P

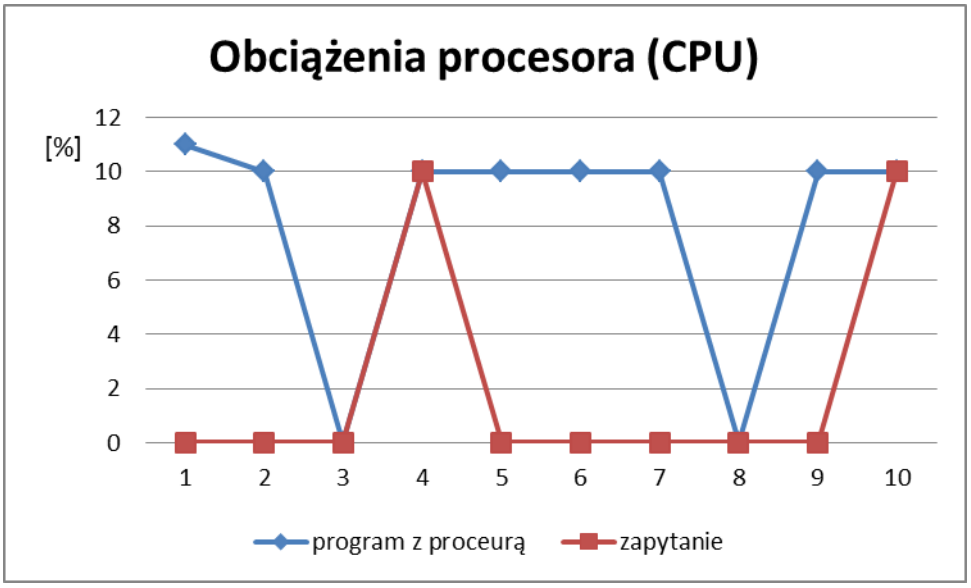
Lp.	Obciążenie procesora [%]	Czas trwania zapytania [ms]
1	0	1
2	0	1
3	0	1
4	10	2
5	0	1
6	0	1
7	0	1
8	0	1
9	0	1
10	10	2
Wartość średnia	**2**	**1,2**
Odchylenie standardowe	4,21637	0,421637

Źródło: opracowanie własne

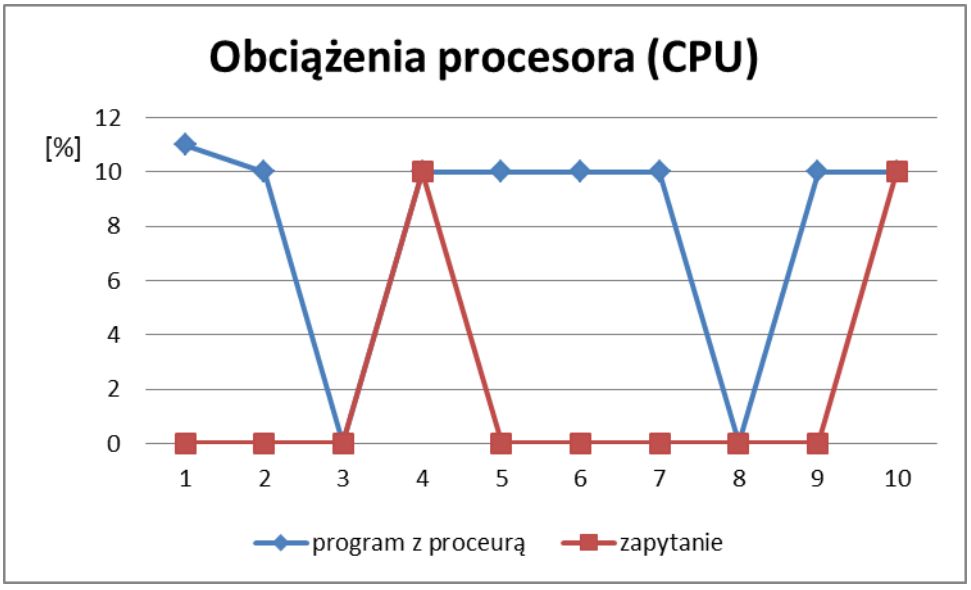
Tabela 12.3. Wyniki pomiarów dla programu z procedurą z listingu 12.R

Lp.	Obciążenie procesora [%]	Czas trwania zapytania [ms]
1	11	7
2	10	9
3	0	7
4	10	6
5	10	10
6	10	9
7	10	7
8	0	8
9	10	7
10	10	8
Wartość średnia	8,1	7,8
Odchylenie standardowe	4,280446	1,229273

Źródło: opracowanie własne



Rys. 12.2. Porównanie obciążenia procesora programów z listingów 12.4 oraz 12.5
Źródło: opracowanie własne



Rys. 12.3. Porównanie czasu wykonania programów z listingów 12.4 oraz 12.5
Źródło: opracowanie własne

Wyniki jasno pokazują, że zastosowanie procedury jest bardziej czasochłonne, a także znacznie obciążają zajętość procesora. Obciążenie to jest szczególnie zauważalne podczas pierwszego użycia, kiedy jest ono dodatkowo zwiększone w związku z kompilacją procedury.

12.4. WYDAJNOŚĆ ZAPYTAŃ I PODZAPYTAŃ

Zapytania można podzielić na zapytania proste oraz złożone, które zawierają podzapytania. Często jednakowy rezultat można otrzymać wykorzystując różnie skonstruowane zapytania. W szczególności zapytania oparte na wielu tabelach mogą zostać przedstawione w postaci prostego (bez podzapytania) zapytania zawierającego klauzulę `JOIN`. Jednak istnieje możliwość zastąpienia tego rozwiązania zapytaniem złożonym, opartym o podzapytania.

Przykład zapytań, których wynik stanowi listę użytkowników, którzy dodali zdjęcia opatrzone słowami kluczowymi: „podróże” lub „kulinaria” przedstawiają listingi 12.6 oraz 12.7. Listing 12.6 przedstawia zapytanie proste, zawierające klauzulę INNER `JOIN`. Listing 12.7 zawiera zapytanie złożone, oparte o kilka podzapytań.

Chociaż wynik obu zapytań jest jednakowy, to ich parametry (czas trwania oraz obciążenie procesora) różnią się. Średni czas wykonywania obu zapytań był równy i wyniósł 2,8 ms. Odchylenia standardowe tych wartości różniły się i wyniosły odpowiednio dla zapytania prostego 1,03 i 0,43 dla zapytania z podzapytaniem. Natomiast średnie obciążenie procesora dla zapytania prostego to 0 (odchylenie standardowe również wyniosło 0), a dla zapytania z podzapytaniem **–** 3 ms (odchylenie standardowe to 4,83). Dokładne czasy oraz obciążenia przedstawiają tabela 12.4 (zapytanie proste) i 12.5 (zapytanie złożone) oraz rysunki 12.4 (zapytanie proste) i 12.5 (zapytanie złożone). Wyniki pokazują, że zapytania proste, bez podzapytań, wykonują się nieco szybciej i ich realizacja obciąża procesor w znacznie mniejszym stopniu.

Listing 12.6. Zapytanie proste

```
SELECT DISTINCT u.id_user, login, name, surname
FROM Users u INNER JOIN Comments c ON u.id_user=c.id_user
INNER JOIN Photos p ON p.id_photo=c.id_photo
INNER JOIN Keyword_to_photos kp ON kp.id_photo=p.id_photo
INNER JOIN Keywords k ON k.id_keyword=kp.id_keyword
WHERE keyword IN ('podróże','kulinaria')
AND last_login_success IS NOT NULL
```

Źródło: opracowanie własne

Listing 12.7. Zapytanie złożone

```
SELECT id_user, login, name, surname
FROM Users
WHERE id_user IN (
SELECT id_user
FROM Comments
WHERE id_photo IN (
SELECT id_photo
FROM Photos
WHERE id_photo IN (
SELECT id_photo
FROM Keyword_to_photos
WHERE id_keyword IN (
SELECT id_keyword
FROM Keywords
WHERE keyword IN ('podróże','kulinaria')))))
AND last_login_success IS NOT NULL
```

Źródło: opracowanie własne

Tabela 12.4. Wyniki pomiarów dla zapytania prostego

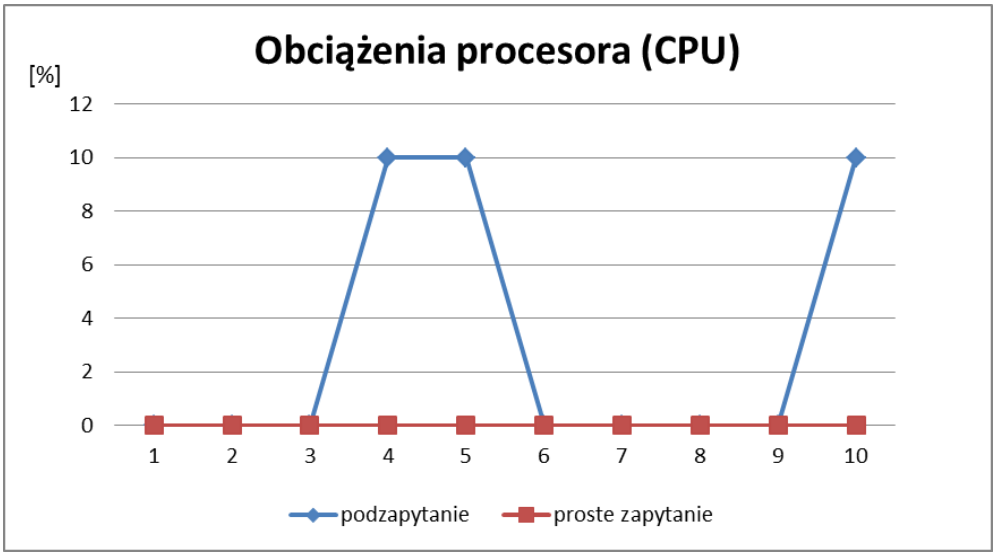
Lp.	Obciążenie procesora [%]	Czas trwania zapytania [ms]
1	0	2
2	0	2
3	0	3
4	0	2
5	0	3
6	0	2
7	0	5
8	0	3
9	0	2
10	0	4
Wartość średnia	**0**	**2,8**
Odchylenie standardowe	**0**	**1,03**

Źródło: opracowanie własne

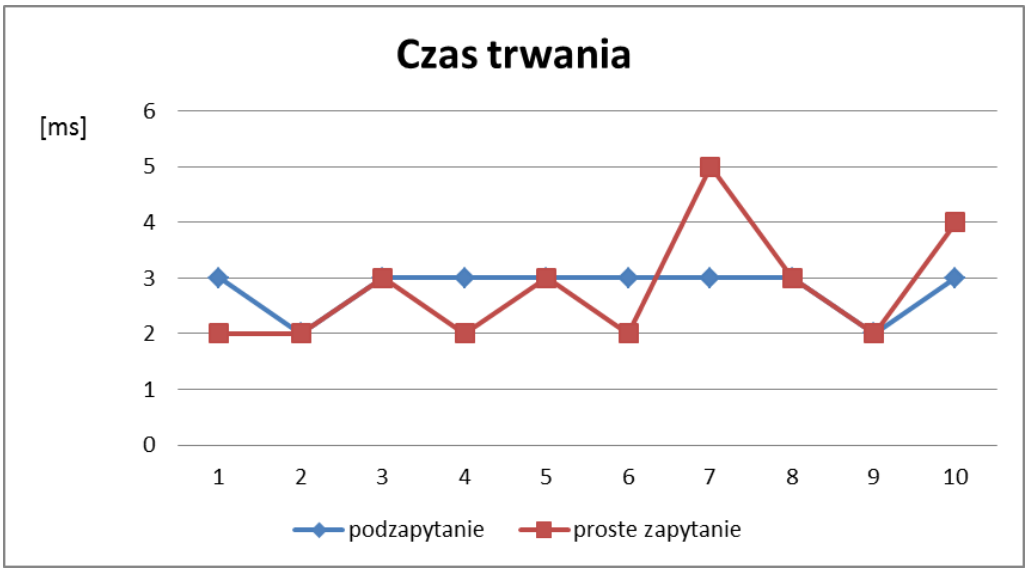
Tabela 12.5. Wyniki pomiarów dla podzapytania

Lp	Obciążenie procesora [%]	Czas trwania zapytania [ms]
1	0	3
2	0	2
3	0	3
4	10	3
5	10	3
6	0	3
7	0	3
8	0	3
9	0	2
10	10	3
Wartość średnia	**3**	**2,8**
Odchylenie standardowe	**4,83**	**0,42**

Źródło: opracowanie własne



Rys. 12.4. Porównanie obciążeń procesora (CPU) z listingów 12.6 oraz 12.7
Źródło: opracowanie własne



Rys. 12.5. Porównanie czasu trwania zapytań z listingów 12.5 oraz 12.7
Źródło: opracowanie własne

12.5. WYDAJNOŚĆ WYBRANYCH POLECEŃ

Złożone zapytania z podzapytaniami również mogą być realizowane na różne sposoby. Ten sam rezultat może być osiągnięty, z wykorzystaniem różnych funkcji oraz klauzul SQL. Przykładem mogą być podzapytania przedstawione na listingach 12.8 – 12.10. Wszystkie zapytania zwracają dane użytkownika, który jako pierwszy ocenił zdjęcie umieszczone w repozytorium. Zapytanie zrealizowano na trzy sposoby, z których każdy wykorzystuje podzapytanie. Sposób pierwszy (Listing 12.8) w podzapytaniu wykorzystuje klauzulę `TOP`. Sposób drugi i trzeci (odpowiednio listingi 12.9 i 12.10) przedstawiają wykorzystanie klauzuli `MIN`, ostatni sposób korzysta dodatkowo z klauzuli `HAVING`.

Najbardziej efektywnym sposobem okazało się zapytanie z Listingu 12.8, którego średnie obciążenie procesora wyniosło 0 (odchylenie standardowe równe jest 0), a średni czas realizacji – 1,8 ms (odchylenie standardowe – 0,63). Drugie pod względem efektywności było zapytanie z klauzulą TOP. Wartość średnia dla obciążenia procesora wyniosła 1 (odchylenie standardowe – 3,16), a wartość średnia dla czasu realizacji – 2,4 ms (odchylenie standardowe – 1,43).

Najmniej efektywne okazało się wykorzystanie klauzuli HAVING. Średnia wartość dla obciążenia procesora to 1 (odchylenie standardowe – 3,16), a średni czas realizacji – 3,2 ms (odchylenie standardowe – 3,91).

Wyniki tych analiz potwierdzają, że warto upraszać tworzone zapytania. Funkcja MIN okazała się wydajniejsza niż klauzula TOP, która wymagała dodatkowo

przeprowadzenia operacji sortowania (z klauzulą ORDER BY). Jednak najmniej efektywne okazało się wykorzystanie klauzuli HAVING, która zawęża wyniki zapytania później, niż klauzula WHERE. Zapytanie z klauzulą HAVING działa więc dłużej, ponieważ wymusza przetwarzanie (w tym wykonanie podzapytania) na większej liczbie danych niż ma to miejsce w przypadku wykorzystania klauzuli WHERE. W tym przypadku wykorzystanie klauzuli HAVING nie było konieczne (ponieważ warunek nie dotyczył funkcji agregującej), więc lepszym rozwiązaniem jest rezygnacja z jej użycia.

Listing 12.8. Zapytanie z klauzulą `TOP`

```
SELECT S.*
FROM (
SELECT Users.id_user, login, name, surname, rate, id_rate
FROM Users INNER JOIN Rates ON Users.id_user=Rates.id_user
WHERE rate_creation_date=(SELECT TOP 1 rate_creation_date FROM
Rates ORDER BY rate_creation_date)) AS S
```

Źródło: opracowanie własne

Listing 12.9. Zapytanie z klauzulą `MIN`

```
SELECT S.*
FROM (
SELECT Users.id_user, login, name, surname, rate, id_rate
FROM Users INNER JOIN Rates ON Users.id_user=Rates.id_user
WHERE rate_creation_date=(SELECT MIN(rate_creation_date) FROM
Rates)) AS S
```

Źródło: opracowanie własne

Listing 12.10. Zapytanie z klauzulą HAVING

```
SELECT S.*
FROM (
SELECT Users.id_user, login, name, surname, rate, id_rate
FROM Users INNER JOIN Rates ON Users.id_user=Rates.id_user
GROUP BY Users.id_user, login, name, surname, rate, id_rate,
rate_creation_date
HAVING rate_creation_date=(SELECT MIN(rate_creation_date) FROM
Rates)) AS S
```

Źródło: opracowanie własne

Tabela 12.6. Wyniki pomiarów dla zapytania z klauzulą TOP

Lp.	Obciążenie procesora [%]	Czas trwania zapytania [ms]
1	0	6
2	0	1
3	0	2
4	10	2
5	0	2
6	0	3
7	0	2
8	0	1
9	0	3
10	0	2
Wartość średnia	**1**	**2,4**
Odchylenie standardowe	**3,16**	**1,43**

Źródło: opracowanie własne

Tabela 12.7. Wyniki pomiarów dla zapytania z klauzulą MIN

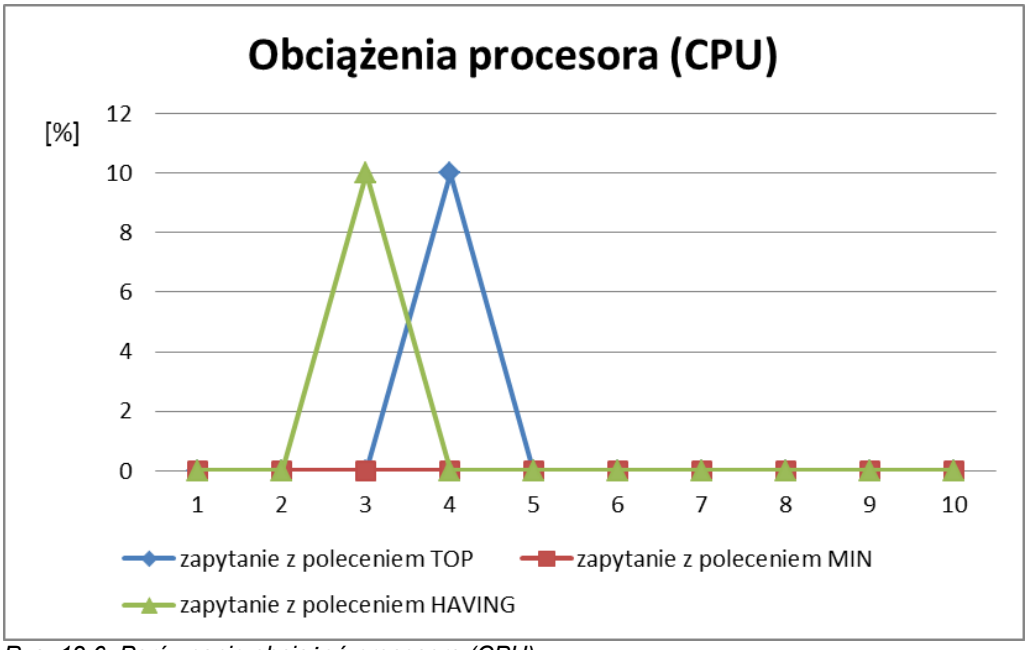
Lp.	Obciążenie procesora [%]	Czas trwania zapytania [ms]
1	0	2
2	0	3
3	0	2
4	0	2
5	0	1
6	0	2
7	0	1
8	0	1
9	0	2
10	0	2
Wartość średnia	**0**	**1,8**
Odchylenie standardowe	**0**	**0,63**

Źródło: opracowanie własne

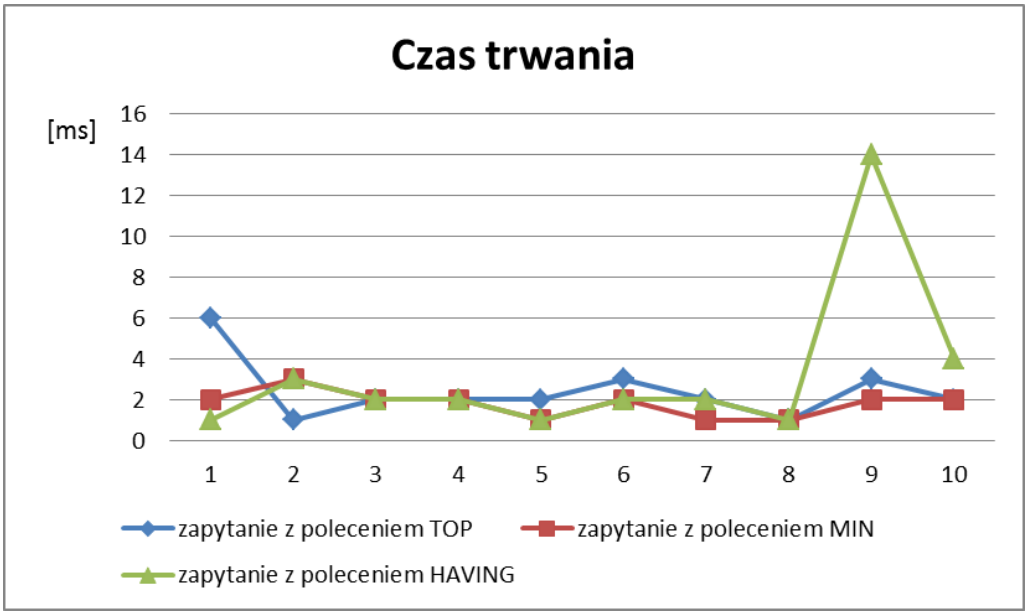
Tabela 12.8. Wyniki pomiarów dla zapytania z klauzulą HAVING

Lp.	Obciążenie procesora [%]	Czas trwania zapytania [ms]
1	0	1
2	0	3
3	10	2
4	0	2
5	0	1
6	0	2
7	0	2
8	0	1
9	0	14
10	0	4
Wartość średnia	**1**	**3,2**
Odchylenie standardowe	**3,16**	**3,91**

Źródło: opracowanie własne



Rys. 12.6. Porównanie obciążeń procesora (CPU)
Źródło: opracowanie własne



Rys. 12.7. Porównanie czasu trwania zapytań
Źródło: opracowanie własne

Analiza porównawcza zajętości procesora oraz czasu trwania programów zostały zilustrowane na rysunkach 12.6 oraz 12.7.

12.6. WYDAJNOŚĆ KURSORÓW

Przeprowadzono porównanie obciążenia procesora oraz czasu trwania programów definiujących i korzystających z kursorów. Na listingach 12.11 oraz 12.12 zostały przedstawione dwa programy które pobierają kursor i analizują na jakim poziomie ocenione zostały zdjęcia w podanym miesiącu. Drugi listing przedstawia analogiczny program, ale z użyciem procedury *OcenyZdjecKursor* z jednym parametrem – numerem miesiąca.

Listing 12.11. Program mierzący parametry wydajnościowe programu oceny zdjęć w wybranym miesiącu

```
DECLARE
@plik nvarchar(30), @ocena int, @id_zd int, @id_uz int,
      @log nvarchar(30),@data date, @miesiac int
SET @miesiac = 1;
DECLARE OcenyZdjec CURSOR FORWARD_ONLY
FOR
 select file_name, rate, r.id_photo, r.id_user,login,
       rate_creation_date
 from Rates r JOIN Users u on r.id_user = u.id_user
 JOIN Photos p on r.id_photo = p.id_photo
 where MONTH(rate_creation_date) = @miesiac
 order by r.id_photo, rate;
OPEN OcenyZdjec;
FETCH NEXT FROM OcenyZdjec
INTO @plik, @ocena, @id_zd, @id_uz, @log, @data;
IF @@FETCH_STATUS <>0
 PRINT 'Brak ocen dodanych w miesiącu'+
       convert(varchar(2),@miesiac)
ELSE
 PRINT 'Oceny zdjęć dodane w miesiącu'+
       convert(varchar(2),@miesiac)
WHILE @@FETCH_STATUS = 0
BEGIN
IF @ocena > 5
 PRINT 'Użytkownik ' + @log + ' ocenił zdjecie – ' + @plik +'
      jako bardzo dobre - ('+ Convert(varchar(2), @ocena)+ ')'
ELSE IF @ocena > 3
 PRINT 'Użytkownik ' + @log + ' ocenił zdjecie – ' + @plik +'
      jako dobre - ('+ Convert(varchar(2), @ocena)+ ')'
ELSE IF @ocena > 2
 PRINT 'Użytkownik ' + @log + ' ocenił zdjecie – ' + @plik +'
      jako średnie - ('+ Convert(varchar(2), @ocena)+ ')'
ELSE
 PRINT 'Użytkownik ' + @log + ' ocenił zdjecie – ' + @plik +'
      jako okropne - ('+ Convert(varchar(2), @ocena)+ ')'
FETCH NEXT FROM OcenyZdjec
INTO @plik, @ocena, @id_zd, @id_uz, @log, @data;
END
CLOSE OcenyZdjec;
DEALLOCATE OcenyZdjec;
```

Źródło: opracowanie własne

Listing 12.12. Program mierzący wydajnościowe parametry programu oceny zdjęć w wybranym miesiącu z użyciem procedury z kursorem

```
DECLARE
@mies int
SET @mies = 1;
EXEC OcenyZdjecKursor @miesiac = @mies
```

Źródło: opracowanie własne

Wyniki uzyskane podczas dziesięciu uruchomień obu programów, związanych z obciążeniem procesora oraz czasem trwania zostały pokazane na rysunkach 12.8 i 12.9 oraz tabelach 12.9 i 12.10, które dodatkowo zawierają policzone średnie wartości oraz odchylenie średnie przedstawionych wyników.

Tabela 12.9. Wyniki pomiarów dla programu z kursorem

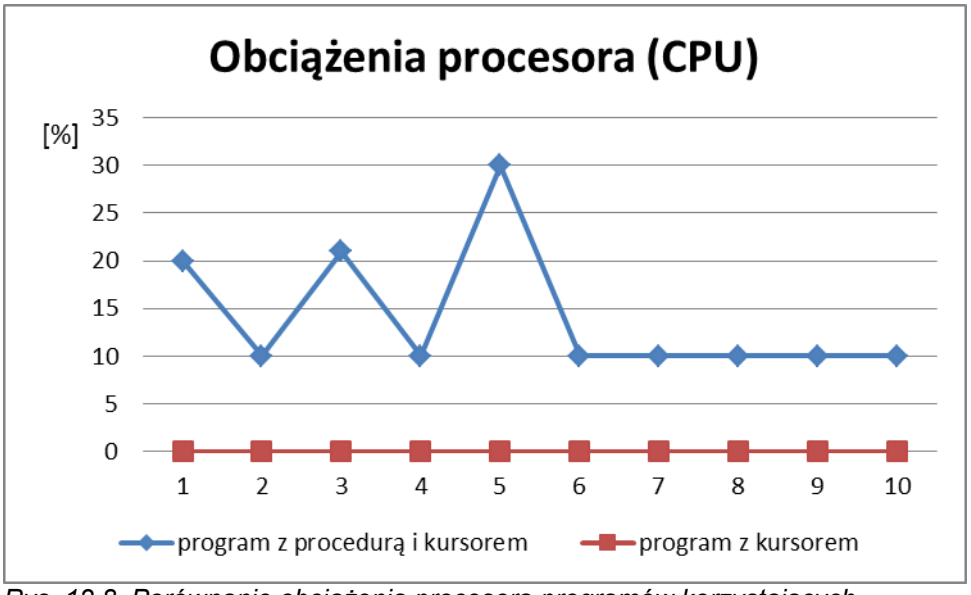
Lp.	Obciążenie procesora [%]	Czas trwania zapytania [ms]
1	0	0
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0
10	0	0
Wartość średnia	**0**	**0**
Odchylenie standardowe	**0**	**0**

Źródło: opracowanie własne

Tabela 12.10. Wyniki pomiarów dla programu z kursorem i procedurą

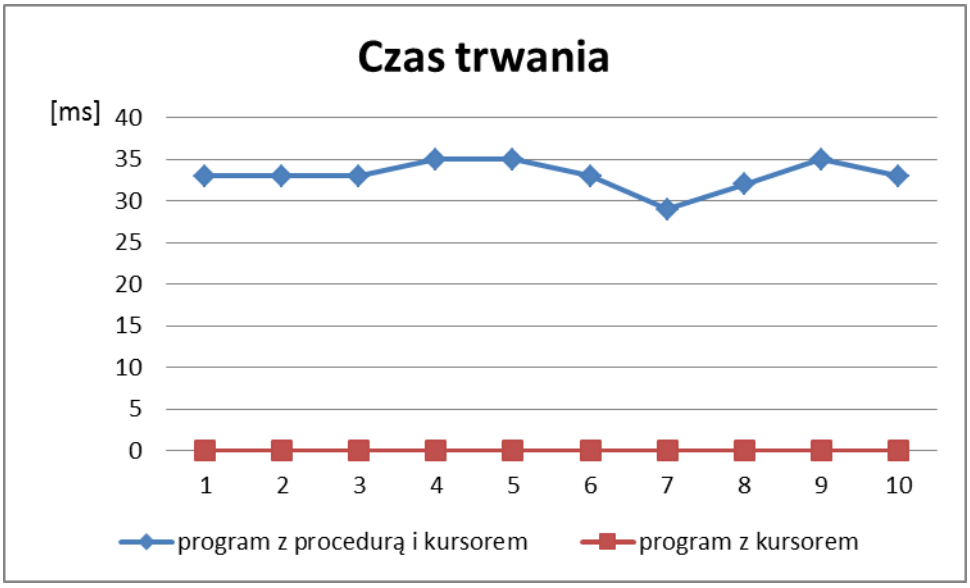
Lp.	Obciążenie procesora [%]	Czas trwania zapytania [ms]
1	20	33
2	10	33
3	21	33
4	10	35
5	30	35
6	10	33
7	10	29
8	10	32
9	10	35
10	10	33
Wartość średnia	14,1	33,1
Odchylenie standardowe	10	33

Źródło: opracowanie własne



Rys. 12.8. Porównanie obciążenia procesora programów korzystających z kursora zarówno z jak i bez procedury.

Źródło: opracowanie własne



Rys. 12.9. Porównanie czasu trwania programów korzystających z kursora zarówno z jak i bez procedury.

Źródło: opracowanie własne

Przedstawione wyniki wskazują, że zastosowanie procedur dodatkowo obciąża procesor i program jest dłużej wykonywany. Mimo tej niedogodności, należy pamiętać, że procedury ułatwiają grupowanie wybranych poleceń, a dzięki temu ułatwiają administrowanie systemem bazodanowym.

12.7. PYTANIA KONTROLNE

1. Podaj polecenia czyszczące cache serwera bazodanowego.
2. Podaj polecenia pomiary czasu wykonania programu T-SQL.
3. W jaki sposób można wykonać pomiar obciążenia procesora wykonania programu T-SQL?

Literatura

DeBetta P., Low G., Whitehorn M. (2008) Introducting Microsoft SQL Server 2008, Redmont, Microsoft Press.

Kalbarczyk W. (2010) T-SQL Nowości dla programistów, http://msdn.microsoft.com/pl-pl/library/t-sql-nowosci-dla-programistow-cz-1.aspx,

Lobel L., Brust A., Forte S. (2009) Programowanie Microsoft SQL Server 2008, APN Promise.

Łojewski Z. (2011) Bazy danych – teoria i praktyka, Lublin, Wydawnictwo UMCS.

Łuszczyk W., (2008) T-SQL Język zapytań bazy danych MS SQL. Kurs dla poczatkujących, http://www.volago.pl/misc/Skrypt.pdf

Microsoft MSDN Library, ALTER FUNCTION (Transact-SQL), http://msdn.microsoft.com/pl-pl/library/ms186967.aspx

Microsoft MSDN Library, ALTER PROCEDURE (Transact-SQL), http://msdn.microsoft.com/en-us/library/ms189762.aspx

Microsoft MSDN Library, Case (Transact-SQL), http://msdn.microsoft.com/en-us/library/ms181765

Microsoft MSDN Library, Cast and Convert (Transact-SQL), http://msdn.microsoft.com/en-us/library/ms187928.aspx

Microsoft MSDN Library, CLOSE (Transact-SQL), http://msdn.microsoft.com/en-us/library/ms175035.aspx

Microsoft MSDN Library, Columnproperty (Transact-SQL),
http://msdn.microsoft.com/en-us/library/ms174968.aspx
Microsoft MSDN Library, CREATE FUNCTION (Transact-SQL),
http://msdn.microsoft.com/en-us/library/ms186755.aspx
Microsoft MSDN Library, CREATE PROCEDURE (Transact-SQL),
http://msdn.microsoft.com/en-us/library/ms187926.aspx
Microsoft MSDN Library, CURSORS (Transact-SQL),
http://msdn.microsoft.com/en-us/library/ms181441.aspx
Microsoft MSDN Library, Data Types (Transact-SQL),
http://msdn.microsoft.com/en-us/library/ms187752.aspx
Microsoft MSDN Library, DBCC DROPCLEANBUFFERS (Transact-SQL),
http://msdn.microsoft.com/en-us/library/ms187762.aspx
Microsoft MSDN Library, DBCC FREEPROCCACHE (Transact-SQL),
http://msdn.microsoft.com/pl-pl/library/ms174283%28v=sql.105%29.aspx
Microsoft MSDN Library, DEALLOCATE (Transact-SQL),
http://msdn.microsoft.com/en-us/library/ms188782.aspx
Microsoft MSDN Library, Declare @local_variable (Transact-SQL),
http://msdn.microsoft.com/en-us/library/ms188927
Microsoft MSDN Library, DECLARE CURSOR (Transact-SQL),
http://msdn.microsoft.com/en-us/library/ms180169.aspx
Microsoft MSDN Library, DROP FUNCTION (Transact-SQL),
http://msdn.microsoft.com/pl-pl/library/ms190290.aspx
Microsoft MSDN Library, DROP PROCEDURE (Transact-SQL),
http://msdn.microsoft.com/en-us/library/ms174969.aspx
Microsoft MSDN Library, Exists (TOP),
http://msdn.microsoft.com/en-us/library/ms189463.aspx
Microsoft MSDN Library, Exists (Transact-SQL),
http://msdn.microsoft.com/en-us/library/ms188336.aspx
Microsoft MSDN Library, FETCH (Transact-SQL),
http://msdn.microsoft.com/en-us/library/ms180152.aspx
MSDN Library, @@FETCH_STATUS (Transact-SQL),
http://msdn.microsoft.com/en-us/library/ms187308.aspx

Microsoft MSDN Library, Getdate (Transact-SQL),
http://msdn.microsoft.com/en-us/library/ms188383
Microsoft MSDN Library, Left (Transact-SQL),
http://msdn.microsoft.com/en-us/library/ms177601
Microsoft MSDN Library, Ltrim (Transact-SQL),
http://msdn.microsoft.com/en-us/library/ms177827.aspx
Microsoft MSDN Library, OPEN (Transact-SQL),
http://msdn.microsoft.com/en-us/library/ms190500.aspx
Microsoft MSDN Library, OPERATORS (Transact-SQL),
http://msdn.microsoft.com/en-US/library/ms174986%28v=sql.90%29.aspx
Microsoft MSDN Library, ORDER BY,
http://msdn.microsoft.com/en-us/library/bb399723.aspx
Microsoft MSDN Library, SELECT Clause (Transact-SQL),
http://msdn.microsoft.com/en-us/library/ms176104.aspx
Microsoft MSDN Library, SELECT @local_variable (Transact-SQL),
http://msdn.microsoft.com/en-us/library/ms187330
Microsoft MSDN Library, SET STATISTICS TIME (Transact-SQL),
http://msdn.microsoft.com/en-us/library/ms190287.aspx
Microsoft MSDN Library, PRINT (Transact-SQL),
http://msdn.microsoft.com/en-us/library/ms176047.aspx
Microsoft MSDN Library, SET @local_variable (Transact-SQL)
http://msdn.microsoft.com/en-us/library/ms189484
Microsoft MSDN Library, String Functions (Transact-SQL),
http://msdn.microsoft.com/en-us/library/ms181984.aspx
Microsoft MSDN Library, Transact-SQL Reference (Database Engine), SQL Server 2008,
http://msdn.microsoft.com/en-us/library/bb510741%28SQL.100%29.aspx
Microsoft MSDN Library, Using Joins,
http://msdn.microsoft.com/en-us/library/ms191472(v=sql.105).aspx
Microsoft MSDN Library, Using Outer Joins,
http://msdn.microsoft.com/en-us/library/ms187518(SQL.105).aspx

Microsoft MSDN Library, Where (Transact-SQL), http://msdn.microsoft.com/en-us/library/ms188047.aspx

Microsoft MSDN Library, While (Transact-SQL), http://msdn.microsoft.com/en-us/library/ms178642.aspx

Microsoft MSDN Library, Writing SQL Queries: Let's Start with the Basics, http://msdn.microsoft.com/en-US/library/bb264565%28v=SQL.90%29.aspx

Nielsen P., White M., Parui U. (2009), Microsoft SQL Server 2008. Bible, Indianapolis, Wiley.

Transact SQL User's Guide, (2005) http://infocenter.sybase.com/help/topic/com.sybase.help.ase_15.0.sqlug/sqlug.pdf

Turley P., Wood D. (2009), Beginning T-SQL with Microsoft SQL Server 2005 and 2008, Indianapolis, Wiley.

Ullman J., Widom J (2011). Podstawowy kurs systemów baz danych, Wydanie III, Helion.

Indeks