KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

EFS
Europejski Fundusz Społeczny

Projekt POKL.04.01.01-00-295/08-00
pt. „Wzmocnienie i rozwój potencjału dydaktycznego w zakresie kształcenia z informatyki zgodnie z potrzebami rynku pracy”.
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego.

Bazy danych

Materiały dydaktyczne do nauki z wykorzystaniem metod i technik kształcenia na odległość

Wersja 1.0

Damian Dudek

Wyższa Szkoła Informatyki i Zarządzania „Copernicus”
we Wrocławiu

Informacje ogólne

Niniejsze materiały e-learning do nauczania zdalnego kursu *Bazy danych* zostały przygotowane w ramach projektu POKL.04.01.01-00-295/08-00 *Wzmocnienie i rozwój potencjału w zakresie kształcenia z informatyki zgodnie z potrzebami rynku pracy*, realizowanego przez Wyższą Szkołę Informatyki i Zarządzania „Copernicus" we Wrocławiu, w ramach Programu Operacyjnego Kapitał Ludzki (zadanie nr 2: *Opracowanie materiałów dydaktycznych i wdrożenie programów kształcenia z wykorzystaniem metod i technik uczenia na odległość, w tym osób niepełnosprawnych*), finansowanego ze środków Europejskiego Funduszu Społecznego oraz Ministerstwa Nauki i Szkolnictwa Wyższego.

Moduł 6
Przetwarzanie danych za pomocą języka T-SQL

Bieżący moduł poświęcony jest metodom przetwarzania danych w bazie relacyjnej za pomocą języka Transact-SQL.

Czego możesz się nauczyć korzystając z tego modułu?

- Jakie operacje są najczęściej wykonywane w bazach transakcyjnych (OLTP).
- Jak wstawiać, modyfikować i usuwać dane z tabel za pomocą poleceń języka SQL.
- Do czego służą obiekty kodu SQL przechowywane na serwerze: widoki, funkcje, procedury przechowywane.
- W jaki sposób tworzy się zapytania do tabel.
- Jak można oprogramować przydatne podsumowania i raporty z danych w tabelach.
- W jaki sposób można pobrać dane połączone z wielu tabel.
- Jak programuje się zaawansowane mechanizmy w języku T-SQL: transakcje, wyzwalacze, dynamiczny kod SQL.

Całkowity czas realizacji materiału niniejszego modułu (samodzielna nauka) wynosi około **7.0 godzin**.

6.1 Wstawianie, modyfikowanie i usuwanie danych

We wcześniejszych modułach omówione zostały metody projektowania struktury relacyjnej bazy danych oraz jej implementowania za pomocą języka SQL – podzbioru DDL. Teraz nadeszła pora, aby poznać polecenia SQL z podzbioru DML, które służą do przetwarzania danych w tabelach. Jak pamiętamy z pierwszego modułu, podstawową grupą operacji modyfikujących dane w bazie transakcyjnej (OLTP) są operacje CRUD (Create, Read, Update, Delete). W języku SQL jest ona realizowana odpowiednio za pomocą poleceń:

- INSERT – wstawianie wierszy do tabeli;
- SELECT – wybieranie (wyszukiwanie) wierszy z tabeli;
- UPDATE – aktualizacja wierszy w tabeli;
- DELETE – kasowanie wierszy.

W niniejszym rozdziale zostaną omówione komendy INSERT, UPDATE i DELETE, a w rozdziale 6.3 – polecenie SELECT.

Wstawianie danych – instrukcja INSERT

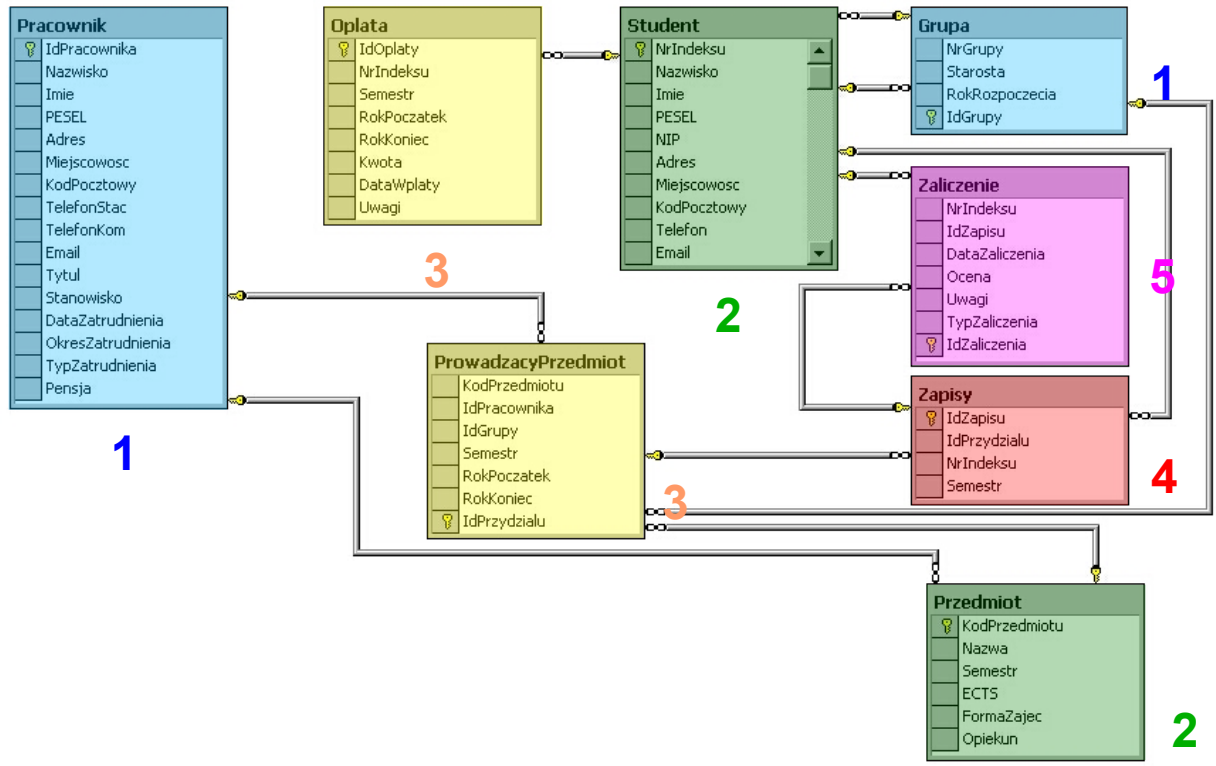
Instrukcja INSERT dodaje nowy wiersz do tabeli. Składnia (najważniejsze elementy):

```
INSERT [ INTO]
{ table_name WITH ( < table_hint_limited > [ ...n ] )
| view_name
| rowset_function_limited
}
{ [ ( column_list ) ]
{ VALUES
( { DEFAULT | NULL | expression } [ ,...n] )
| derived_table
| execute_statement
}
}
```

Instrukcja INSERT – przykłady

```
-- Dodawanie jednego wiersza do tabeli [Przedmiot].
INSERT Przedmiot
```

```
(KodPrzedmiotu, Nazwa, Semestr, ECTS, FormaZajec, Opiekun)
VALUES ('INF407W', 'Bazy danych', 4, 4, '20000', 1)
-- Wstawianie wielu wierszy z zastąpieniem klauzuli "VALUES"
-- przez zapytanie "SELECT": „Wstaw to tabeli [Przedmiot_Archiwum]
-- wszystkie przedmioty zawierające w nazwie słowo „baz”.
INSERT Przedmiot_Archiwum
(KodPrzedmiotu, Nazwa, Semestr, ECTS, FormaZajec, Opiekun)
SELECT KodPrzedmiotu, Nazwa, Semestr, ECTS, FormaZajec, Opiekun
FROM Przedmiot
WHERE Nazwa LIKE '%' + 'baz' + '%';
GO
```



Rys. 6.1. Kolejność wstawiania danych do tabel w bazie „Uczelnia”.

Kolejność wstawiania danych

- Ze względu na relacje między tabelami (ograniczenia FK) wstawianie wierszy nie może odbywać się w dowolnej kolejności, lecz tak, aby zachowana była integralność referencyjna.
- Najpierw wstawiane są wiersze do tabel, do których są odwołania z innych

tabel (przez FK). Odwrotna kolejność powoduje błędy.

• Właściwy porządek wstawiania dla bazy [Uczelnia_Wroclaw]. Jeśli na jednym poziomie jest więcej, niż jedna tabela, oznacza to, że są one wzajemnie niezależne i można do nich wstawiać wiersze naprzemiennie.

1) Tabele: [Pracownik], [Grupa];

2) Tabele: [Student], [Przedmiot];

3) Tabele: [Oplata], [ProwadzacyPrzedmiot];

4) Tabela: [Zapisy];

5) Tabela: [Zaliczenie].

Kolejność wstawiania danych

Zrzut ekranowy przedstawia bazę danych „Wyższa uczelnia techniczna", wersja 1.0.

Modyfikowanie danych – instrukcja UPDATE

Instrukcja UPDATE służy do aktualizacji wartości kolumn w tabeli. Składnia (najważniejsze elementy):

```
UPDATE
{
table_name WITH ( < table_hint_limited > [ ...n ] )
| view_name
| rowset_function_limited
}
SET
{ column_name = { expression | DEFAULT | NULL }
| @variable = expression
| @variable = column = expression } [ ,...n ]
{ { [ FROM { < table_source > } [ ,...n ] ]
[ WHERE < search_condition > ] }
```

Instrukcja UPDATE – przykłady

```
-- Ustanowienie starosty (student z [NrIndeksu] = 3) w grupie '5DB inf'.
UPDATE Grupa
SET Starosta = 3
WHERE NrGrupy = '5DB inf'
GO
-- Modyfikacja kilku pól w jednej instrukcji UPDATE.
UPDATE Pracownik
SET Adres = 'ul. Nizinna 84/9',
TelefonStac = '+48-76-790-18-90',
Stanowisko = 'wykładowca'
WHERE IdPracownika = 10;
GO
```

Usuwanie danych – DELETE, TRUNCATE TABLE, DROP TABLE

W języku T-SQL dane z tabeli można usunąć na trzy sposoby.

• DELETE – wiersze są usuwane z tabeli z możliwością określenia warunku (opcjonalna klauzula WHERE):

– operacja jest rejestrowana w dzienniku transakcji;

– licznik pola autonumerycznego (IDENTITY) pozostaje niezmieniony;

• TRUNCATE TABLE – usuwa wszystkie wiersze z tabeli, dając wynik podobny do instrukcji DELETE bez klauzuli WHERE, przy czym:

– strony przechowujące dane są zwalniane bez rejestrowania operacji w dzienniku transakcji (czyli bezpowrotnie);

– operacja jest znacznie szybsza od DELETE, jeśli trzeba usunąć wszystkie wiersze;

– licznik pola autonumerycznego (IDENTITY) jest ustawiany na wartość początkową (SEED).

• DROP TABLE – trwale usuwa ze schematu bazy danych określoną tabelę, a wraz z nią wszystkie jej dane (rozwiązanie niezalecane po wdrożeniu bazy).

Instrukcja DELETE – usuwanie

Instrukcja DELETE służy do kasowania wierszy z tabeli.

Sama tabela pozostaje w schemacie bazy danych, nawet jeśli jest pusta.

Składnia (najważniejsze elementy):

```
DELETE
[ FROM ]
{ table_name WITH ( < table_hint_limited > [ ...n ] )
| view_name
| rowset_function_limited
}
[ FROM { < table_source > } [ ,...n ] ]
[ WHERE < search_condition > ]
```

DELETE i TRUNCATE TABLE Przykłady

```
-- Kasowanie wszystkich wierszy z tabeli [Zaliczenie].
DELETE Zaliczenie
GO
-- Usuwanie zaliczeń wystawionych w lutym 2005 r.
DELETE Zaliczenie
WHERE DataZaliczenia BETWEEN '2005-02-01' AND '2005-02-28'
GO
-- Szybkie usuwanie wszystkich wierszy w tabeli [Zaliczenie] z ustawieniem
-- licznika IDENTITY na wartość początkową (SEED).
TRUNCATE TABLE Zaliczenie
GO
```

Kolejność usuwania danych

• Podobnie, jak w przypadku instrukcji INSERT, kasowanie danych w bazie musi się odbywać w określonej kolejności, ze względu na relacje między tabelami (ograniczenia FK).

• Właściwy porządek usuwania wierszy z tabel jest odwrotny do tego, który był przy wstawianiu.

• W bazie [Uczelnia_Wroclaw]:

1) Tabela: [Zaliczenie];

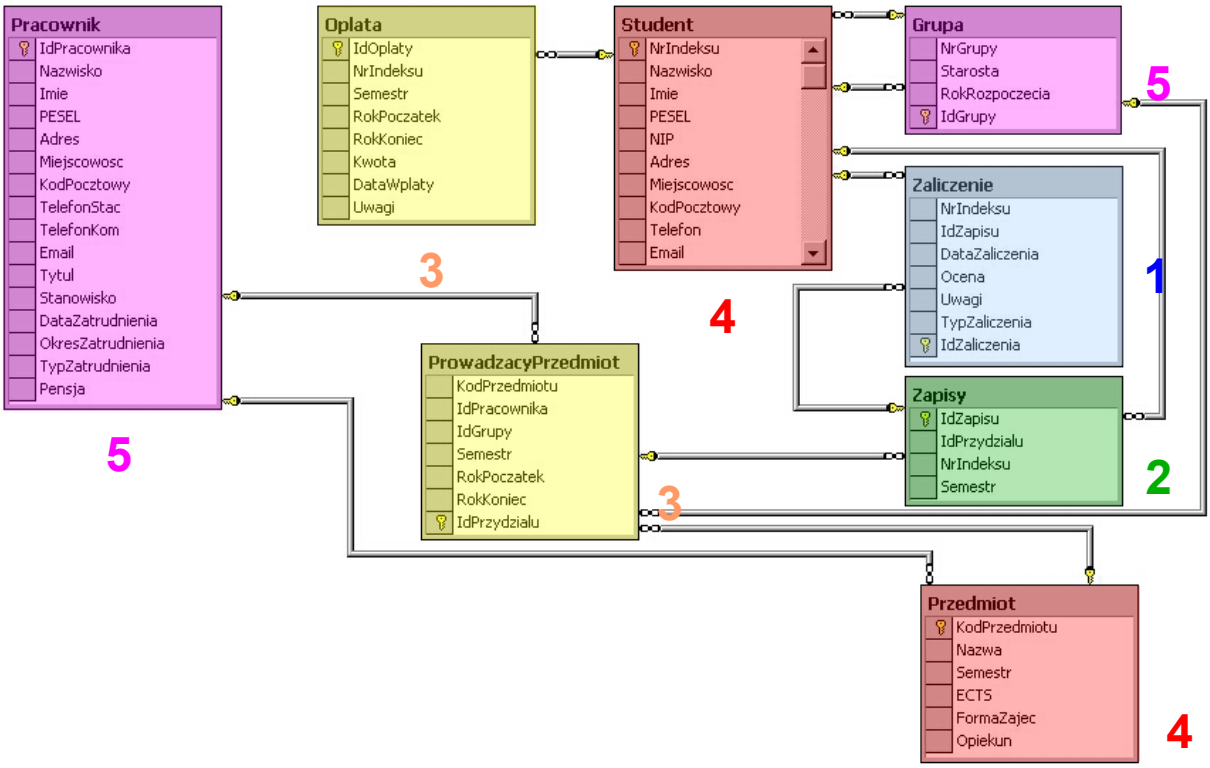
2) Tabela: [Zapisy];

3) Tabele: [Oplata], [ProwadzacyPrzedmiot];

4) Tabele: [Student], [Przedmiot];

5) Tabele: [Pracownik], [Grupa].

Zrzut ekranowy przedstawia bazę danych „Wyższa uczelnia techniczna”, wersja 1.0.



Rys. 6.2. Kolejność usuwania danych z tabel w bazie „Uczelnia”.

6.2 Wyszukiwanie danych z bazy

1. Polecenie SELECT

Instrukcja SELECT zwraca (domyślnie) wszystkie wiersze spełniające warunek wyszukiwania. Składnia – ważniejsze elementy:

```
SELECT [ ALL | DISTINCT ] [ TOP [ PERCENT ] [ WITH TIES ] columns
[ AS column_heading ]
[ INTO new_table_name ]
FROM table_source [ AS table_alias ]
[ INNER | {{ LEFT | RIGHT | FULL } [ OUTER ] } | [ CROSS ] ] | JOIN
table2
[ [ AS ] table_alias ]
[ WHERE search_condition ]
[ GROUP BY [ ALL group_by_expression ] ]
[ HAVING search_condition ]
[ ORDER BY order_expression [ ASC | DESC ] ]
```

Instrukcja SELECT Parametry

- **`ALL`** – zwrócone mogą być wiersze z powtórzeniami wartości (ustawienie domyślne);
- **`DISTINCT`** – zwracane są wyłącznie różne wiersze;
- **`TOP n [ PERCENT ]`** – zwracane jest n pierwszych wierszy (pierwsze n procent wierszy);
- **`WITH TIES`** – zwrócone może być więcej, niż n pierwszych wierszy, jeśli mają one wartość sortowanego pola taką samą, jak ostatni, n-ty wiersz; wymagana jest klauzula ORDER BY;
- **`columns`** – kolumny, które mają być zwrócone w wyniku; symbol "*" zwraca wszystkie kolumny;
- **`AS column_heading`** – przy wyświetlaniu wyników nazwa kolumny jest

zmieniona na „column heading”;

• **INTO** – tworzona jest nowa tabela, do której kopiowane są wyniki zapytania;

• **FROM** – określna tabele (jedną lub więcej), z których mają być zwrócone wiersze;

• **INNER | {LEFT, RIGHT, FULL} OUTER | CROSS JOIN** – złączenie z inną tabelą – odpowiednio: wewnętrzne, zewnętrzne (lewo- i prawostronne, pełne), krzyżowe;

• **WHERE** – określa wyrażenie ograniczające zbiór zwracanych wierszy;

• **search_condition** – wyrażenie, które musi być spełnione, aby dany wiersz był zwrócony.

• **GROUP BY** – określa kolumny (lub wyrażenia nieagregujące), względem których mają być grupowane wiersze w wyniku;

• **HAVING** – klauzula definiująca agregację lub warunek zapytania przy grupowaniu;

• **ORDER BY** – definiuje kolumny, według których ma być sortowany wynikowy zbiór wierszy;

• **ASC | DESC** – porządek sortowania – odpowiednio: rosnący (domyślnie) albo malejący.

SELECT: klauzula WHERE

Klauzula WHERE pozwala na zdefiniowanie warunku, który ogranicza wynikowy zbiór wierszy. Dopuszczalne są następujące operatory:

• **=, < > (różny), <, >, >=, <=** – zwyczajne operatory porównania;

• **BETWEEN wyrażenie1 AND wyrażenie2** – operator “pomiędzy” (łącznie z końcami przedziału);

• **IN | NOT IN (element1,element2,...,elementN)** – wartość kolumny musi (albo nie może) należeć do podanego zbioru;

• **LIKE wyrażenie** – pozwala na porównania przybliżone; można je definiować

za pomocą znaków wieloznacznych (ang. wildcards):

– "%"d – dowolny znak lub cyfra;

– "-"d – dowolny pojedynczy znak;

– "[ ]" - dowolny znak umieszczony w nawiasach.

SELECT...FROM...WHERE Przykład

```
-- Proste wyszukiwanie przedmiotów, które zawierają w nazwie podany wzorzec.
CREATE PROC Przedmiot_szukanie
@Par_Nazwa varchar(128) = ' ' -- Wzorzec, który musi wystąpić w nazwie przedmiotu.
AS
BEGIN -- Kolumny [KodPrzedmiotu] i [Forma] mają zmienione nagłówki przez AS.
SELECT KodPrzedmiotu AS [Kod], Nazwa, Semestr, ECTS,
FormaZajec AS [Forma]
FROM Przedmiot
WHERE Nazwa LIKE '%' + @Par_Nazwa + '%'
ORDER BY KodPrzedmiotu ASC -- Sortowanie rosnące według pola [KodPrzedmiotu].
END
GO
-- Przykładowe wywołanie.
EXEC Przedmiot_szukanie 'sieci';
GO
```

SELECT: kolumny wyliczeniowe

W instrukcji SELECT można podawać, oprócz nazw pól, także wyrażenia obliczane na podstawie pól i funkcji.

```
-- Zapytanie: "zwróć tytuł, imię i nazwisko pracowników – jako pojedyncze pole,
-- posortowane po nazwisku".
SELECT Tytul + Imie + Nazwisko AS [Pracownik]
FROM Pracownik
ORDER BY Nazwisko ASC
GO
-- Zapytanie: "zwróć numer indeksu, imię, nazwisko studenta i liczbę miesięcy,
-- które upłynęły od daty jego zapisania (staż)".
```

```
-- Staż jest obliczany na podstawie pola [DataZapisania] i bieżącej daty
systemowej.

SELECT NrIndeksu, Imie, Nazwisko,

DATEDIFF(Month, DataZapisania, GETDATE( )) AS [Staz na uczelni]

FROM Student

ORDER BY [Staz na uczelni] DESC, Nazwisko ASC

GO
```

Klauzule GROUP BY i HAVING

• Klauzula `GROUP BY` umożliwia utworzenie podsumowań dla wybranych grup tabeli.

• Jeżeli wraz z `GROUP BY` ma być użyta klauzula `WHERE`, to musi być ona umieszczona przed grupowaniem. Klauzula `WHERE` nie może zawierać funkcji agregujących.

• Każde pole wymienione na liście `SELECT`, nie będące wyrażeniem agregującym, musi być umieszczone także w klauzuli `GROUP BY`, albo listy pól po `SELECT` i `GROUP BY` muszą być identyczne.

• Zalecane jest używanie wraz z `GROUP BY` klauzuli `ORDER BY`, aby grupowane wiersze zostały zwrócone w podanej kolejności. W przeciwnym razie kolejność ta nie jest określona.

• Klauzula `HAVING` pozwala na ograniczenie liczby grupowanych wierszy na podstawie podanego warunku, który może zawierać funkcje agregujące.

GROUP BY i HAVING Przykład

```
-- Procedura zliczająca studentów, mieszkających w poszczególnych miejscowościach.
-- Pod uwagę brane są tylko te miejscowości, których nazwa zawiera podany wzorzec
-- i w których mieszka co najmniej zadana liczba studentów.
CREATE PROCEDURE Student_zlicz_miejsc
@Par_Miejscowosc varchar(40) = ' ', -- Wzorzec nazwy miejscowości.
@Par_MinStud int = 0 -- Minimalna liczba studentów.
AS
```

```
BEGIN
SELECT Miejscowosc, COUNT(*) AS [Liczba studentow]
FROM Student
WHERE Miejscowosc LIKE '%' + @Par_Miejscowosc + '%'
GROUP BY Miejscowosc
HAVING COUNT(*) >= @Par_MinStud -- Warunek minimalnej liczby studentów.
ORDER BY [Liczba studentow] DESC -- Sortowanie malejące po liczbie studentów.
END
GO
```

Złączenia tabel

Złączenia służą do przedstawiania połączonych danych z więcej, niż jednej tabeli**.**

• Połączenie opiera się na porównaniu wartości kolumn w obu tabelach,

które często opowiadają relacji kluczy głównych i obcych.

• Wyróżniane są następujące rodzaje złączeń:

– wewnętrzne (ang. inner join);

– własne (ang. self join) – odmiana złączenia wewnętrznego;

– zewnętrzne (ang. outer join);

– krzyżowe (ang. cross join).

• Są dwa rodzaje składni określającej złączenie:

– warunek złączenia w klauzuli `WHERE` (składnia stara, niezalecana);

– warunek złączenia w klauzuli `FROM` (nowa, zalecana składnia,

zgodna ze standardem ANSI-92).

Złączenie tabel jako operacja algebry relacyjnej

Złączeniem (wewnętrznym) tabel R i S jest tabela T, w której wiersze mają połączone kolumny z tabel R i S. W tabeli T są tylko te wiersze, w których nastąpiło dopasowanie wartości określonych kolumn z obu tabel.

Tabela *R*

Imie	Nazwisko	KodPrzedmiotu
Jan	Kowalski	INF507
Andrzej	Jabłoński	INF517
Krzysztof	Nowak	INF517
Leszek	Morawski	INF517

N — 1

Tabela *S*

KodPrzedmiotu	Nazwa	ECTS
INF407	Bazy danych	4
INF507	Sieciowe bazy danych	4
INF517	Grafika komputerowa	5

Złączone tabele *R* i *S*

Imie	Nazwisko	KodPrzedmiotu	Nazwa	ECTS
Jan	Kowalski	INF507	Sieciowe bazy danych	4
Andrzej	Jabłoński	INF517	Grafika komputerowa	5
Krzysztof	Nowak	INF517	Grafika komputerowa	5
Leszek	Morawski	INF517	Grafika komputerowa	5

Rys. 6.3. Złączenie naturalne dwóch tabel jako operacja algebry relacyjnej.

```
-- Odpowiednik w języku SQL (nowa składnia zgodna z ANSI-92):
SELECT Imie, Nazwisko, KodPrzedmiotu, Nazwa, ECTS
FROM R INNER JOIN S ON R.KodPrzedmiotu = S.KodPrzedmiotu
```

Złączenia tabel składnia nowa i stara

```
-- Oba poniższe zapytania zwracają identyczny wynik:
-- „zwróć dane przedmiotów i nazwiska ich opiekunów merytorycznych”.
-- Nowa składnia, zgodna z ANSI-92 (zalecana).
-- Warunek złączenia jest podany w klazuli FROM.
SELECT KodPrzedmiotu AS [Kod], Nazwa, Nazwisko AS [Opiekun merytoryczny]
FROM Przedmiot INNER JOIN Pracownik
ON Przedmiot.Opiekun = Pracownik.IdPracownika;
GO
-- Składnia stara (niezalecana).
-- Warunek złączenia jest podany w klazuli WHERE.
SELECT KodPrzedmiotu AS [Kod], Nazwa, Nazwisko AS [Opiekun merytoryczny]
FROM Przedmiot, Pracownik
WHERE Przedmiot.Opiekun = Pracownik.IdPracownika
GO
```

Zalety nowej składni (ANSI)

• Przejrzystość i zrozumiałość – zwłaszcza przy złączeniach wielu tabel jednocześnie.

• Zapewnienie podwójnych złączeń zewnętrznych, których nie obsługuje stara składnia.

• Możliwość użycia poleceń dla optymalizatora zapytań, określających sposób złączenia tabel. Opcja ta nie jest dostępna w starej składni.

• W kolejnych wersjach MS SQL Server stara składnia może nie być obsługiwana.

Złączenie wewnętrzne: INNER JOIN

Jest to najczęściej stosowane złączenie. Wiersze zwracane w wyniku złączenia wewnętrznego mają taką samą wartość określonej kolumny w obu łączonych tabelach.

```
-- Zapytanie: "zwróć studentów z podanej grupy".
CREATE PROCEDURE Student_Grupa_szukanie
@Par_NrGrupy varchar(7) = ' '
AS
BEGIN
SELECT NrIndeksu AS [Numer indeksu], Imie, Nazwisko, NrGrupy AS [Grupa]
FROM Student INNER JOIN Grupa -- Nazwy tabel i rodzaj ich złączenia.
ON Student.IdGrupy = Grupa.IdGrupy -- Opis kolumn łączących.
-- Nie wszystkie kolumny łączące nie muszą występować na liście SELECT.
WHERE NrGrupy LIKE '%' + @Par_NrGrupy + '%'
ORDER BY NrGrupy ASC, Nazwisko ASC;
END
GO
```

Złączenie własne: szczególny przypadek INNER JOIN

Złączenie własne (ang. self join) jest odmianą złączenia wewnętrznego, w której dana tabela jest łączona z samą sobą.

```
-- Zapytanie: "zwróć studentów, którzy mają takie samo nazwisko".
CREATE PROCEDURE Student_Nazwisko
@Par_Nazwisko varchar(7) = ' '
AS
BEGIN
SELECT stud1.NrIndeksu, stud1.Imie, stud1.Nazwisko,
stud2.NrIndeksu, stud2.Imie, stud2.Nazwisko, stud1.Miejscowosc
FROM Student stud1 INNER JOIN Student stud2 -- Wykorzystywane są aliasy.
ON stud1.Nazwisko = stud2.Nazwisko -- Dopasowywane kolumny.
WHERE (stud1.Nazwisko LIKE '%' + @Par_Nazwisko + '%')
AND (stud1.NrIndeksu < stud2.NrIndeksu) -- Warunek zapobiegający powtórzeniom.
ORDER BY stud1.Nazwisko ASC
END;
GO
```

Złączenie zewnętrzne: OUTER JOIN

Złączenie zewnętrzne zwraca wszystkie wiersze z tabeli, która została określona jako zewnętrzna (`LEFT OUTER`, `RIGHT OUTER`, `FULL OUTER`), nawet jeśli wartości łączonych kolumn nie są równe.

• Wiersze tabeli zewnętrznej, dla których nie istnieje dopasowanej wartości z tabeli wewnętrznej, zawierają w kolumnie łączonej wartość `NULL`.

• Złączenie lewostronne (`LEFT OUTER JOIN`) i prawostronne (`RIGHT OUTER JOIN`) nie różnią się między sobą poza kolejnością specyfikacji tabeli zewnętrznej i wewnętrznej.

• Złączenie zewnętrzne pełne (`FULL OUTER JOIN`) zwraca wszystkie odpowiadające sobie wiersze z łączonych tabel, a następnie wiersze, dla których dopasowanie nie zachodzi.

Złączenie zewnętrzne: OUTER JOIN – przykład

```
-- Zwracanie grup i ich starostów (jeśli są ustanowieni).
CREATE PROC Grupa_szukanie
@Par_RokRozpoczecia smallint = 1990 -- Rok rozpoczęcia studiów przez grupę.
```

```
AS
BEGIN
-- Imię i nazwisko starosty grupy są sklejane w pojedynczy ciąg znaków
-- po prawostronnym obcięciu spacji przez funkcję RTRIM( ).
SELECT NrGrupy AS [Nazwa grupy], RokRozpoczecia AS [Rok rozpoczecia],
RTRIM (Imie) + ' ' + RTRIM (Nazwisko) AS [Starosta], NrIndeksu AS [Nr indeksu]
-- Lewostronne złączenie zewnętrzne zwraca wszystkie grupy spełniające warunek.
-- WHERE, nawet jeśli nie mają one starostów (wówczas są to pola NULL).
FROM Grupa LEFT OUTER JOIN Student
ON Grupa.Starosta = Student.NrIndeksu
WHERE RokRozpoczecia >= @Par_RokRozpoczecia
END;
GO
```

Złączenie krzyżowe: CROSS JOIN

Wynikiem złączenia krzyżowego są wiersze, zawierające wszystkie możliwe kombinacje wartości określonych kolumn (iloczyn kartezjański tabel).

Liczba zwracanych wierszy może być bardzo duża.

```
-- Złączenie krzyżowe.
-- Zapytanie: "wypisz wszystkie kombinacje imion i nazwisk studentów".
CREATE PROCEDURE Student_Imie_Nazwisko
AS
BEGIN
SELECT DISTINCT stud1.Imie, stud2.Nazwisko
FROM Student stud1 CROSS JOIN Student stud2
ORDER BY stud2.Nazwisko, stud1.Imie
END;
GO
```

Podzapytania

Podzapytanie jest zapytaniem SQL, które występuje wewnątrz innego zapytania i może być użyte w miejscu wyrażenia.

• Wynikiem podzapytania może być pojedynczy wiersz lub kolumna tabeli.

• Wyniki podzapytania i złączenia są często takie same. Złączenie jest jednak bardziej wydajne niż podzapytanie. Wyjątkiem jest sytuacja, w której mają być usunięte powtarzające się wiersze (`DISTINCT`). Wówczas podzapytanie użyte z `NOT EXISTS` działa szybciej.

• Podzapytania są konieczne w przypadku zapytań, które:

– zawierają warunki nieistnienia lub niewchodzenia w skład (`NOT EXISTS`, `NOT IN`);

– wymagają użycia funkcji agregującej w klauzuli `WHERE`.

Podzapytania – przykłady

```
-- Poniższe zapytanie można realizować alternatywnie za pomocą złączeń
-- (jest ono tak zaimplementowane w procedurze Student_Grupa_szukanie).
-- Zapytanie: "zwróć studentów z podanej grupy".
CREATE PROCEDURE Student_Grupa_podzapytanie
@Par_NrGrupy varchar(7) = ''
AS
BEGIN
SELECT NrIndeksu AS [Numer indeksu], Imie, Nazwisko
FROM Student
WHERE IdGrupy IN (SELECT IdGrupy
FROM Grupa
WHERE NrGrupy LIKE '%' + @Par_NrGrupy + '%')
ORDER BY Nazwisko ASC, Imie ASC
END
GO
-- Poniższego zapytania NIE MOŻNA realizować alternatywnie za pomocą złączenia
-- wewnętrznego, ponieważ odwołuje się do warunków nieistnienia.
-- Zapytanie: "zwróć studentów, którzy nie wnieśli żadnej opłaty od podanej daty".
CREATE PROCEDURE Student_Oplata_podzapytanie
@Par_DataWplaty datetime
AS
```

```
BEGIN
SELECT DISTINCT NrIndeksu AS [Numer indeksu], Imie, Nazwisko
FROM Student
WHERE NrIndeksu NOT IN (SELECT NrIndeksu
FROM Oplata
WHERE DataWplaty >= @Par_DataWplaty)
ORDER BY Nazwisko ASC, Imie ASC
END
GO
-- Poniższego zapytania NIE MOŻNA realizować alternatywnie
-- za pomocą złączeń, ponieważ wykorzystuje ono funkcję agregującą
-- w klauzuli WHERE.
-- Zapytanie: "zwróć pracowników, których wynagrodzenie jest niższe
-- od średniej płacy wszystkich pracowników".
CREATE PROCEDURE Pracownik_Placa_podzapytanie
AS
BEGIN
SELECT Tytul, Imie, Nazwisko, Pensja
FROM Pracownik
WHERE Pensja < (SELECT AVG(Pensja)
FROM Pracownik)
ORDER BY Pensja ASC, Nazwisko ASC, Imie ASC
END
GO
```

Podzapytania jako tabele pochodne

W zapytaniu można zastosować tabelę pochodną, która jest zapytaniem zagnieżdżonym, zawartym w klauzuli `FROM`.

• Do tabeli pochodnej można odwoływać się przez alias i można traktować ją jak normalną tabelę lub jak dynamiczny widok.

• Podzapytanie może zawierać agegacje (np. `COUNT`), grupowania (`GROUP BY`) i sumy wierszy (`UNION`).

Podzapytania jako tabele pochodne – przykład

```
-- Zapytanie:
-- "wypisz wszystkie kombinacje imion i nazwisk studentów i pracowników".
CREATE PROCEDURE Student_Prac_podzapytanie
AS
BEGIN
SELECT DISTINCT stud1.Imie, stud2.Nazwisko
FROM (SELECT Imie FROM Student
UNION
SELECT Imie FROM Pracownik) stud1
CROSS JOIN
(SELECT Nazwisko FROM Student
UNION
SELECT Nazwisko FROM Pracownik) stud2
ORDER BY stud2.Nazwisko, stud1.Imie
END
GO
```

6.3 Obiekty kodu SQL przechowywane na serwerze

Główna idea – możliwe korzyści w stosunku do zapytań ad hoc; widoki; procedury przechowywane – zalety (bezpieczeństwo, wydajność, zintegrowanie kodu logiki aplikacyjnej) i wady (ograniczenia języka T-SQL). Przykłady – baza Uczelnia – EM / QA 2000). Procedury w architekturze klient-serwer – przykłady PHP / ASP.NET.

Programowanie procedur – budowa procedury, parametryzacja (INPUT/OUTPUT), zwracanie danych, implementacja typowych procedur CRUD dla poleceń INSERT/UPDATE/DELETE/SELECT; przykłady (zwłaszcza zapytań SELECT dla różnych potrzeb informacyjnych użytkownika końcowego).

1. Widoki

Widok (ang. view, nazywany także „perspektywą") – jest zapytaniem SQL, przechowywanym w bazie danych, w postaci obiektu. Stanowi on swego rodzaju „okno na dane", znajdujące się w bazie.

• Cele stosowania widoków:

– dostarczenie użytkownikowi tylko takiej informacji, która jest dla niego przydatna i zrozumiała (np. dane zagregowane lub złączone z wielu tabel, nie zawierające identyfikatorów autonumerycznych);

– zwiększenie wydajności - serwer przechowuje kompilowany plan wykonania dla widoku (nieindeksowanego), dzięki czemu jego uruchamianie jest szybsze od wywoływania zwykłego zapytania SQL;

– kontrola dostępu do danych – różni użytkownicy mogą mieć prawo uruchamiania różnych widoków w ramach tej samej bazy danych.

• Widoki nie zajmują miejsca na dysku, gdyż nie są fizycznie przechowywane w bazie, lecz tworzone na bieżąco w formie tabel wirtualnych.

• Za pomocą widoku można nie tylko wyświetlać dane, ale także wstawiać je do tabel, modyfikować i usuwać (z pewnymi ograniczeniami).

Zasady tworzenia widoków

• Widok jest zawsze oparty na poleceniu `SELECT`, które wybiera dane z jednej lub kilku tabel (albo widoków). Polecenie to podajemy po słowie `AS`.

• Wyrażenie `SELECT` w definicji widoku nie może zawierać:

– odwołania do tabeli tymczasowej lub zmiennej tabelarycznej;

– odwołania do więcej niż 1024 kolumn;

– klauzuli `ORDER BY`, chyba że wyrażenie `SELECT` zawiera także klauzulę `TOP`;

– słowa kluczowego `INTO`;

– klauzul `COMPUTE` i `COMPUTE BY`.

• Widoki muszą być tworzone w bieżącej bazie danych.

W poleceniu `CREATE VIEW` nie można określić nazwy bazy.

Widoki jako obiekty w bazie

Widoki są obiektami kodu T-SQL, trwale przechowywanymi w bazie danych. Można je przeglądać w programach „Enterprise Manager" oraz „Query Analyzer", w węzłach „Views". Można także wyświetlać informację o widokach z poziomu T-SQL, za pomocą poleceń `sp_tables` i `sp_helptext <view_name>`.

Tworzenie i modyfikowanie widoku – składnia

Polecenia `CREATE VIEW` i `ALTER VIEW` są praktycznie identyczne, ponieważ każda modyfikacja widoku polega na zamianie jego dotychczasowej zawartości na nową, określoną w poleceniu `ALTER VIEW`.

```
CREATE / ALTER VIEW [ < database_name > . ] [ < owner > . ] view_name
[ ( column [ ,...n ] ) ]
[ WITH < view_attribute > [ ,...n ] ]
AS
select_statement
[ WITH CHECK OPTION ]
< view_attribute > ::=
{ ENCRYPTION | SCHEMABINDING | VIEW_METADATA }
```

T-SQL: Tworzenie widoku Przykłady

```
-- Prosty widok wyświetlający przedmioty (bez opiekunów).
CREATE VIEW V_Przedmiot_dane
AS -- Niektóre z wybieranych pól mają zmienioną nazwę – nagłówek kolumny.
```

```
SELECT KodPrzedmiotu AS [Kod przedmiotu], Nazwa,
FormaZajec AS [Forma zajec]
FROM Przedmiot
GO
-- Modyfikacja widoku polega na podaniu nowej zawartości jego kodu T-SQL.
ALTER VIEW V_Przedmiot_dane
AS -- Dodatkowo mają być wyświetlane pola [Semestr] i [ECTS].
SELECT KodPrzedmiotu AS [Kod przedmiotu], Nazwa, Semestr, ECTS,
FormaZajec AS [Forma zajec]
FROM Przedmiot
GO
```

Zwracanie danych z widoku

- Widok może być traktowany jak wirtualna tabela, której wiersze nie są zapisywane trwale w bazie. Zwrócenie danych z widoku odbywa się przez polecenie "`SELECT...`".
- Jeżeli chcemy wyświetlić tylko niektóre kolumny, ich nazwy muszą być zgodne z nazwami kolumn zdefiniowanymi w widoku, a nie w tabelach źródłowych.

```
SELECT [Kod przedmiotu], Nazwa
FROM V_Przedmiot_dane
GO
```

Wstawianie danych przez widok

Za pomocą widoku można nie tylko wyświetlać dane, ale też wstawiać je trwale do bazy. Jest to możliwe, jeżeli widok zawiera wszystkie pola wymagane w tabeli (czyli `NOT NULL`).

```
-- W widoku [V_Przedmiot_dane] są wszystkie pola obowiązkowe, nie ma zaś
-- pola [Opiekun] z dopuszczalną wartością NULL. Możemy zatem wykorzystać
-- ten widok do wstawiania danych.
INSERT V_Przedmiot_dane
-- Ważne jest, aby nazwy pól były zgodne z definicją widoku, a nie docelowej
tabeli.
```

```
([Kod przedmiotu], Nazwa, Semestr, ECTS, [Forma zajec])
VALUES
('INF708W', 'Windows 2', 7, 4, '20000')
GO
```

Modyfikowanie i usuwanie danych przez widok

Widok może także służyć do modyfikowania i usuwania wierszy na identycznych zasadach, jak w przypadku zwykłych tabel.

```
-- Modyfikowanie danych przez widok.
UPDATE V_Przedmiot_dane
SET ECTS = 5
WHERE [Kod przedmiotu] = 'INF708W'
GO
-- Usuwanie danych przez widok.
DELETE V_Przedmiot_dane
WHERE Semestr = 7
GO
```

Wizualne projektowanie widoków

Program „SSMS" zawiera funkcję wygodnego, wizualnego projektowania widoków. Odpowiedni kod T-SQL jest tworzony automatycznie. Można też na bieżąco sprawdzać wyniki zwracane przez widok.

2. Procedury przechowywane

Procedura przechowywana (ang. stored procedure) – jest to jedno lub więcej poleceń języka SQL, przechowywanym w bazie danych w postaci wykonywalnego obiektu.

Cechy procedur przechowywanych:

– można je wywoływać interakcyjnie z poziomu aplikacji klienta,

a także z innej procedury przechowywanej lub wyzwalanej;

– procedury mogą pobierać i zwracać parametry, co zwiększa ich użyteczność

i elastyczność (np. w porównaniu z widokami);

– mogą zwracać zbiór wyników (np. wierszy z tabeli) i kod wyjścia;

– procedury są przechowywane na serwerze w formie skompilowanej – razem z planem wykonania (execution plan).

Procedury jako obiekty w bazie

Procedury przechowywane są obiektami kodu T-SQL, trwale przechowywanymi w bazie danych. Można je przeglądać w programach „Enterprise Manager" oraz „Query Analyzer", w węzłach „Stored procedures". Można także wyświetlać informację o procedurach za pomocą polecenia `sp_stored_procedures`.

Sposoby zwracania danych z procedury przechowywanej

• Procedura przechowywana może zwracać dane na cztery sposoby:

– zbiór wierszy (ang. record set), będący wynikiem wykonania polecenia `SELECT`;

– parametry wyjściowe (`OUTPUT`), zwracające wartości (np. integer, char) albo wskaźnik na kursor (zmienna typu cursor);

– kod powrotu (return code, return value) – zawsze jako liczba całkowita; może służyć do wychwytywania i obsługi błędów;

– globalny kursor, do którego można się odwoływać poza procedurą.

Procedury przechowywane - zalety

• gwarancja integralności danych:

– kodowanie logiki aplikacji w pojedynczym miejscu, co ułatwia zarządzanie regułami biznesowymi i zachowanie ich jednolitości;

– jeśli aplikacja klienta może zmieniać dane wyłącznie za pośrednictwem procedur, jest mniejsze ryzyko wprowadzenia modyfikacji naruszających spójność danych;

– ograniczenie błędów programistycznych przy tworzeniu aplikacji wielowarstwowych

– łatwiejsze wykonywanie złożonych operacji, przekazywanie mniejszej ilości informacji pomiędzy warstwami;

– zautomatyzowanie złożonych transakcji o dużym znaczeniu dla integralności danych;

• modularny charakter programowania – kod SQL podzielony jest na mniejsze fragmenty, które są łatwiejsze do zarządzania;

• łatwiejsze utrzymanie i modyfikacja aplikacji wielowarstwowej – przejrzyste oddzielenie bazy danych od warstwy dostępu do danych;

• wzrost bezpieczeństwa danych – łatwiejsza ochrona przed atakami (np. przed tzw. wstrzyknięciem kodu SQL – ang. SQL injection – zob. poz. nr 1 w spisie literatury);

• ograniczenie ruchu w sieci – procedury (nawet te złożone z bardzo wielu poleceń SQL) są wywoływane zdalnie za pomocą pojedynczego polecenia z parametrami;

• wzrost wydajności przy powtarzających się wywołaniach – procedury na serwerze są optymalizowane (kompilowane przy pierwszym uruchomieniu i przechowywane potem razem z gotowym planem wykonania);

• ograniczenie dostępu do tabel tylko do zdefiniowanych czynności – aplikacja klienta może mieć uprawnienia tylko do procedur przechowywanych, a nie bezpośrednio do tabel.

Procedury przechowywane - wady

• Transact-SQL jest językiem o ograniczonym możliwościach – może on być niewystarczający do oprogramowania bardzo złożonych operacji; sytuacja ta ulega jednak zmianie – na przykład w najnowszej wersji MS SQL Server 2005, do implementacji procedur przechowywanych, oprócz standardowego T-SQL, można wykorzystywać także wysoko-poziomowe, obiektowe języki platformy MS .NET (np. C#, VB);

• gorsza integracja ze środowiskami programistycznymi – nie wszystkie narzędzia do tworzenia aplikacji wielowarstwowych pozwalają na kontrolę wersji i usuwanie błędów (ang. debugging) na poziomie procedur przechowywanych (chociaż np. MS Visual Studio .NET ma tę możliwość);

• mała przenośność – mimo istnienia specyfikacji tworzenia procedur

przechowywanych w standardzie ANSI SQL 99, przez większość producentów systemów baz danych nie jest ona w pełni przestrzegana; dlatego kod procedur przechowywanych najczęściej nie daje się bezpośrednio przenosić np. z bazy MS SQL Server do Oracle.

T-SQL: Tworzenie i modyfikowanie procedury – składnia

Podobnie, jak w przypadku widoków, składnia polecenia tworzenia (`CREATE`) i modyfikowania (`ALTER`) procedury przechowywanej jest identyczna, ponieważ każda modyfikacja procedury polega na zamianie jego dotychczasowej zawartości na nową, określoną w poleceniu `ALTER PROCEDURE`.

```
CREATE / ALTER PROC [ EDURE ] procedure_name [ ; number ]
[ { @parameter data_type }
[ VARYING ] [ = default ] [ OUTPUT ]
] [ ,...n ]
[ WITH -- Opcja WITH ENCRYPTION służy do zaszyfrowania kodu T-SQL.
{ RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION } ]
[ FOR REPLICATION ]
AS sql_statement [ ...n ]
```

Tworzenie procedury przechowywanej – przykład

```
-- Dodawanie nowego przedmiotu.
CREATE PROC Przedmiot_wstawianie -- Unikalna nazwa procedury.
@Par_KodPrzedmiotu char(7), -- Lista parametrów.
@Par_Nazwa char(128),
@Par_Semestr tinyint,
@Par_ECTS tinyint = 4, -- Wartość domyślna parametru.
@Par_FormaZajec char(10),
@Par_Opiekun int
AS -- Po słowie „AS” następuje ciało procedury – właściwy ciąg poleceń T-SQL.
INSERT Przedmiot
(KodPrzedmiotu, Nazwa, Semestr, ECTS, FormaZajec, Opiekun)
VALUES -- Zamiast stałych wykorzystywane są wartości parametrów procedury.
```

```
(@Par_KodPrzedmiotu, @Par_Nazwa, @Par_Semestr, @Par_ECTS,
@Par_FormaZajec, @Par_Opiekun)
GO
```

Przykład wywołania procedury przechowywanej

```
-- Do wywołania procedury używamy słowa EXECUTE (albo EXEC), po którym
-- podajemy parametry i ich wartości. Pomaga to uniknąć pomyłek.
EXECUTE Przedmiot_wstawianie
@Par_KodPrzedmiotu = 'INF407W',
@Par_Nazwa = 'Bazy danych',
@Par_Semestr = 4,
@Par_ECTS = DEFAULT, -- Wymuszenie wartości domyślnej parametru.
@Par_FormaZajec = '20000',
@Par_Opiekun = 1
GO
-- Można także stosować skróconą składnię – podajemy tylko wartości parametrów,
-- koniecznie w takiej kolejności, w jakiej są zadeklarowane w definicji procedury.
-- Taki zapis jest krótszy i wygodniejszy, ale bardziej podatny na błędy.
EXEC Przedmiot_wstawianie 'INF507W', 'Sieciowe bazy danych', 5, 4, '10000', 1
GO
```

Procedura z parametrami wyjściowymi – przykład

```
CREATE PROC Przedmiot_edycja -- Procedura zmienia dane przedmiotu.
@Par_KodPrzedmiotu char(7), -- Parametr służący do identyfikacji przedmiotu.
@Par_Nazwa char(128), -- Pozostałe parametry przekazują nowe wartości pól.
@Par_Semestr tinyint, -- Konwencja nazewnicza parametrów wejściowych
@Par_ECTS tinyint, -- „@Par_<nazwa>”, a wyjściowych – „@Out_<nazwa>”.
@Par_FormaZajec char(10),
@Par_Opiekun int,
@Out_LiczbaWierszy int OUTPUT -- Parametr wyjściowy.
AS
UPDATE Przedmiot
SET Nazwa = @Par_Nazwa, Semestr = @Par_Semestr, ECTS = @Par_ECTS,
FormaZajec = @Par_FormaZajec, Opiekun = @Par_Opiekun
```

```
WHERE KodPrzedmiotu = @Par_KodPrzedmiotu
-- Wartość parametru wyjściowego to liczba zmodyfikowanych wierszy.
SELECT @Out_LiczbaWierszy = @@ROWCOUNT
GO
```

Przykład wywołania procedury z parametrami wyjściowymi

```
-- Wyłączamy wyświetlanie standardowego komunikatu "N row(s) affected".
-- To ustawienie normalnie obowiązuje dla bieżącego połączenia.
SET NOCOUNT ON
-- Deklarujemy zmienną do pobrania wartości parametru wyjściowego.
DECLARE @Var_Wiersze int
-- Wywołanie procedury. Po zmiennej, podanej jako parametr wyjściowy,
-- musi być umieszczone słowo "OUTPUT".
EXEC Przedmiot_edycja 'INF407W', 'Bazy danych', 5, 4, '20000', 1,
@Var_Wiersze OUTPUT
-- Wyświetlenie komunikatu tekstowego. Funkcja CONVERT dokonuje konwersji
-- z jednego typu danych na inny (tutaj z int do varchar(3)).
PRINT ('Zmieniono: ' + CONVERT(varchar(3), @Var_Wiersze) + ' wiersz(y).')
-- Przywracamy wyświetlanie komunikatu "N row(s) affected".
SET NOCOUNT OFF
GO
```

6.4 Zaawansowane programowanie w języku T-SQL

Dogłębna znajomość języka Transact-SQL może przynieść programiście baz danych wiele korzyści. Niektóre z nich zostały wymienione poniżej.

• Zaawansowane mechanizmy języka T-SQL pozwalają programiście na implementowanie

wydajnego kodu logiki aplikacyjnej na poziomie bazy danych.

• Zarządzanie serwerem i bazami danych może być szybsze z poziomu języka

T-SQL, niż z konsoli wizualnych. Na przykład aby „Enterprise Manager" mógł

wyświetlić graficznie informację o bazach danych albo kontach logowania, musi

on przetworzyć szereg procedur systemowych i widoków na tabelach systemowych,

a następnie załadować wyniki do obiektów interfejsu użytkownika. To często

powoduje wolniejsze działanie, w stosunku do zwykłej konsoli SQL (takiej, jak

„Query Analyzer").

• Programy w języku T-SQL pozwalają na automatyzację pracy administratora

(np. wygodne nadawanie i odbieranie uprawnień), szczególnie przy złożonych

i powtarzających się zadaniach.

• Jeżeli nie mamy dostępu do narzędzi komercyjnych (np. w niskobudżetowym

projekcie), w języku T-SQL możemy administrować serwerem baz danych takim,

jak MSDE2000, za pomocą bezpłatnej konsoli MS SQL Web Data Administrator

(oba darmowe produkty dostępne na stronie: www.microsoft.com/downloads).

Administrowanie serwerem MS SQL Server za darmo?

Nawet jeśli nasz zleceniodawca nie sfinansuje nam wygodnych narzędzi do implementacji

i zarządzania bazami danych, i tak możemy opracować profesjonalny projekt,

posługując się bezpłatnym oprogramowaniem. Warunek: dobra znajomość technologii.

2. Wbudowane funkcje MS SQL Server

• Język Transact-SQL w MS SQL Server posiada duży zestaw

wbudowanych funkcji, które zwiększają możliwości programowanego kodu.

• Funkcje można podzielić na 3 główne grupy:

– funkcje zestawu wierszy (ang. rowset functions) – zwracają obiekt, który jest odpowiednikiem wskaźnika na tabelę;

– funkcje agregujące (ang. aggregate functions) – operują na zbiorze wierszy, ale zwracają pojedynczą wartość;

– funkcje skalarne (ang. scalar functions) – pobierają i zwracają pojedynczą wartość (ich szczegółowe zestawienie i opis znajduje się w Books Online):

• funkcje konfiguracji (configuration);

• funkcje obsługi kursorów (cursor);

• funkcje daty i czasu (date and time);

• funkcje matematyczne (mathematical) ;

• funkcje metadanych (metadata) ;

• funkcje kontroli dostępu (security);

• funkcje ciągów znakowych (string);

• funkcje systemowe (system) ;

• funkcje informacji o systemie (system statistical);

• funkcje Text i Image (text and image).

Funkcje agregujące

Wykonują operacje na wybranej kolumnie w tabeli lub zbiorze wierszy.

Zwracają pojedynczą wartość liczbową. Są często wykorzystywane w zapytaniach z poleceniem `SELECT`. Wybrane funkcje:

• `AVG ( [ ALL | DISTINCT ] expression )` – średnia wartość wszystkich wartości wyrażenia „expression”;

• `BINARY_CHECKSUM ( * | expression [ ,...n ] )` – binarna suma kontrolna obliczana dla wiersza lub listy wyrażeń; może być użyta do wykrywania zmian w wierszu;

• `COUNT ( { [ ALL | DISTINCT ] expression ] | * } )` – liczba wartości w wyrażeniu „expression” z pominięciem wartości „NULL” (chyba, że podano „*”).

• `MAX ( [ ALL | DISTINCT ] expression )` – wartość maksymalna wyrażenia.

• `MIN ( [ ALL | DISTINCT ] expression )` – wartość minimalna wyrażenia.

• `SUM ( [ ALL | DISTINCT ] expression )` – suma wartości wyrażenia; „`ALL`” (domyślnie) - sumuje wszystkie wartości; „`DISTINCT`” – tylko różne.

Funkcje daty i czasu

Ułatwiają operacje na zmiennych typu datetime lub datepart (część daty).

Wybrane funkcje:

• `DATEADD (datepart, int, date)` – dodaje „`int`” razy „`datepart`” do daty „`date`”;

• `DATENAME (datepart, date)` – zwraca podaną część daty „`datepart`” jako łańcuch znaków (dla miesiąca i dnia tygodnia jest to nazwa);

• `DATEPART (datepart, date)` – zwraca podaną część daty jako liczbę całkowitą;

• `DAY (date)` – zwraca dzień miesiąca dla podanej daty „`date`”;

• `GETDATE ( )` – bieżąca data i czas;

• `GETUTDATE ( )` – bieżąca data i czas Greenwich (GMT);

• `MONTH (date)` – zwraca miesiąć jako liczbę całkowitą;

• `YEAR (date)` – zwraca rok jako liczbę całkowitą.

Funkcje matematyczne

Wykonują obliczenia dla podanych argumentów i zwracają wynik.

Wybrane funkcje:

• `ABS (numeric)` – wartość bezwzględna podanej liczby;

• `DEGREES (numeric)` – zamienia radiany na stopnie;

• `EXP (float)` – funkcja eksponencjalna;

• `FLOOR (numeric)` – największa liczba całkowita, mniejsza lub równa podanej liczbie „`numeric`”;

• `RADIANS (numeric)` – zamienia stopnie na radiany;

• `RAND ([seed])` – liczba losowa z przedziału [0;1], o wartości początkowej „`seed`”;

• `ROUND (numeric, length, prec)` – zaokrąglenie wartości liczby do podanej długości „`length`” i precyzji „`prec`”.

• `SIN (float)` – sinus kąta podanego w radianach.

Funkcje ciągów znakowych

Umożliwiają wykonywanie operacji na ciągach znaków – łańcuchach.

Wybrane funkcje:

• `CHAR (int)` – zwraca znak ASCII odpowiadający podanej liczbie;

• `LOWER (char)` – zamienia podany łańcuch na małe litery (lower case);

• `REPLACE (char1, char2, char 3)` – wszystkie łańcuchy „`char1`", znalezione w łańcuchu „`char2`", są zastępowane przez „`char3`";

• `REVERSE (char)` – odwraca podany łańcuch;

• `RTRIM (char)` – zwraca łańcuch bez spacji końcowych;

• `STR (float, [length, [decimal]])` – zwraca reprezentację łańcuchową liczby `float` o długości „`length`" i z liczbą „`decimal`" miejsc po przecinku. Domyślne wartości – odpowiednio: 10 i 0;

• `SUBSTRING (char, start, length)` – zwraca fragment o długości „`length`" z łańcucha „`char`", począwszy od pozycji „`start`";

• `UPPER (char)` – zamienia podany łańcuch na duże litery (`upper case`).

Funkcje systemowe

Pomagają uzyskać informacje o ustawieniach i opcjach SQL Servera.

Wybrane funkcje:

• `CONVERT (datatype [(length)], expression, style)` – przekształca wyrażenie „`expression`" na podany typ danych (dla typów „`float`" oraz „`datetime`" parametr „`style`" określa sposób formatowania);

• `HOST_NAME ( )` – nazwa stacji roboczej klienta;

• `IDENT_CURRENT ('table_name')` – ostatnio wygenerowana wartość `IDENTITY` dla podanej tabeli;

• `ISDATE (char)` – zwraca wartość „1", jeśli podany łańcuch zawiera poprawną datę („0" w przeciwnym wypadku);

• `ISNULL (expression, value)` – zwraca wartość „`value`", jeśli podane wyrażenie

ma wartość „NULL” („0” w przeciwnym wypadku);

• SERVERPROPERTY ('property') – wartość podanej cechy „property” serwera, zwracana jako „sql_variant”.

Funkcje systemowe bez argumentów

Wybrane funkcje – w nawiasach podane są inne funkcje systemowe o takim samym działaniu:

• CURRENT_TIMESTAMP (GETDATE( )) – bieżąca data i czas;

• CURRENT_USER (USER_NAME( )) – nazwa aktualnego użytkownika;

• SESSION_USER (USER_NAME( )) – nazwa aktualnego użytkownika;

• SYSTEM_USER (SUSER_NAME( )) – nazwa aktualnego użytkownika;

• USER (USER_NAME( )) – nazwa aktualnego użytkownika.

Funkcje metadanych

Pozwalają na uzyskanie informacji o tabelach, kluczach, indeksach itp.

Wybrane funkcje:

• COL_LENGTH (table, column) – długość kolumny „column” w tabeli;

• COL_NAME (table_id, column_id) – nazwa kolumny o podanym identyfikatorze;

• COLUMNPROPERTY (table_id, column, property) – zwraca informacje o kolumnie „column” w tabeli o identyfikatorze „table_id”;

• DATABASEPROPERTY(database_name, property) – zwraca wartość właściwości „property” dla bazy „database_name”);

• DB_ID ([db_name]) – zwraca identyfikator bazy o podanej nazwie;

• DB_NAME ([db_id]) – zwraca nazwę bazy o podanym identifikatorze;

• OBJECT_ID ([object_name]) – zwraca identyfikator obiektu o podanej nazwie.

Funkcje kontroli dostępu

Umożliwiają uzyskiwanie informacji o użytkownikach i rolach.

Wybrane funkcje:

• IS_MEMBER (group | role) – zwraca wartość „1”, jeśli bieżący użytkownik jest członkiem podanej grupy NT lub roli SQL Servera („0” w przeciwnym wypadku);

- `SUSER_SID (login)` – identyfikator SID podanego użytkownika;
- `SUSER_SNAME ([login_id])` – nazwa użytkownika o podanym identyfikatorze.

3. Systemowe procedury przechowywane

Systemowe procedury przechowywane

• MS SQL Server 2000 posiada szereg systemowych i rozszerzonych procedur przechowywanych, które, w połączeniu z poleceniami języka Transact-SQL (przede wszystkim DDL), pozwalają na wykonywanie wszystkich funkcji administracyjnych, w tym także tych dostępnych w konsolach graficznych (np. Enterprise Manager, Query Analyzer).

• Systemowe procedury przechowywane (ang. system stored procedures) są obiektami kodu T-SQL, realizującymi szeroką gamę funkcji zarządzania serwerem i bazami danych.

• Rozszerzone procedury przechowywane są, zarejestrowanymi w SQL Serverze, skompilowanymi bibliotekami dynamicznymi DLL (ang. Dynamic Link Libraries), implementowanymi w języku C/C++, które pozwalają na wywoływanie z poziomu kodu T-SQL zaawansowanych funkcji zarządzania środowiskiem SQL Servera. Użytkownik ma możliwość tworzenia i włączania do serwera SQL własnych procedur rozszerzonych.

• Procedury obsługi Active Directory (ang. Active Directory procedures) – są używane do rejestrowania instancji SQL Servera oraz baz danych w MS 2000 Active Directory.

Procedury systemowe - podział

• Procedury katalogu (ang. catalog procedures) – udostępniają funkcje słownika danych (ang. data dictionary), pozwalając na listowanie baz danych, obiektów w bazach i ich własności.

• Procedury obsługi kursorów (ang. cursor procedures) – służą do zarządzania kursorami.

• Procedury planu utrzymania bazy (ang. Database Maintenance Plan procedures) – są wykorzystywane do zarządzania automatycznymi zadaniami utrzymania i optymalizacji baz danych.

• Procedury zapytań rozproszonych (ang. distributed queries procedures) – pozwalają na zarządzanie zdalnymi źródłami danych.

• Procedury wyszukiwania pełnotekstowego (ang. Full-Text Search procedures) – służą do zarządzania indeksami pełnotekstowymi.

• Procedury wysyłania dziennika transakcji (ang. log shipping procedures) – są używane do administrowania automatyczną synchronizacją transakcji między bazami danych (np. w celu utworzenia serwera zapasowego – ang. backup server).

• Procedury automatyzacji OLE (ang. OLE Automation procedures) – pozwalają na udostępnianie obiektów OLE z poziomu kodu T-SQL, w celu uzyskania dodatkowych funkcji.

• Procedury replikacji (ang. replication procedures) – służą do zarządzania procesem powielania danych między serwerami.

• Procedury systemu bezpieczeństwa (ang. security procedures) – są używane do administrowania zabezpieczeniami serwera i baz danych.

• Procedury modułu SQL Mail (ang. SQL Mail procedures) – udostępniają funkcje obsługi poczty elektronicznej z poziomu SQL Servera.

• Procedury programu SQL Profiler (ang. SQL Profiler procedures) – są wykorzystywane do zarządzania śledzeniem zdarzeń w programie MS SQL Profiler.

• Procedury agenta SQL (ang. SQL Server Agent procedures) – pozwalają na zarządzanie zadaniami wykonywanymi automatycznie.

• Procedury systemu (ang. system procedures) – służą do ogólnego administrowania SQL Serverem.

• Procedury asystenta sieci WWW (ang. Web Assistant procedures) – udostępniają funkcje automatycznego tworzenia dokumentów HTML

na podstawie danych pobranych z bazy.

• Procedury XML (ang. XML procedures) – są wykorzystywane do zarządzania tekstem w języku XML (ang. Extensible Markup Language).

• Procedury rozszerzone (ang. general extended procedures) – pozwalają na dostęp z poziomu języka T-SQL do zewnętrznych bibliotek DLL, które rozszerzają zakres funkcji SQL Servera.

Systemowe procedury przechowywane – przykłady

• `sp_configure` – wyświetla ustawienia serwera i pozwala na ich zmianę;

• `sp_attach_db` – przyłącza bazę do serwera (pliki *.mdf, *.ldf, *.ndf);

• `sp_detach_db` – odłącza bazę z serwera (bez kasowania jej plików);

• `sp_databases` – podaje listę nazw i rozmiarów wszystkich baz danych w bieżącej instancji serwera;

• `sp_helpdb` – wyświetla informację o podanej bazie danych;

• `sp_help` – zwraca informacje o wybranych obiekcie lub o wszystkich obiektach podanej bazy danych;

• `sp_tables` – zwraca listę wszystkich tabel i widoków w bieżącej bazie danych;

• `sp_columns` – wyświetla informacje o polach wybranej tabeli lub widoku;

• `sp_helpconstraint` – zwraca listę ograniczeń podanej tabeli;

• `sp_stored_procedures` – wyświetla listę wszystkich procedur w bazie danych;

• `sp_helptrigger` – podaje listę wyzwalaczy, zdefiniowanych w określonej tabeli;

• `sp_helptext` – wyświetla pełny kod T-SQL widoku, procedury lub wyzwalacza.

4. Walidacja danych.

Walidacja danych (ang data validation) jest procesem sprawdzania poprawności danych, które są wprowadzane do bazy, względem zdefiniowanych więzów integralności. Celem walidacji nie zawsze jest tylko odrzucenie błędnych danych, wprowadzanych przez użytkownika. W niektórych przypadkach wynikiem procesu sprawdzania danych może być także ich automatyczna korekta (np. zamiana pierwszych liter imion i nazwisk na kapitaliki), co może zwiększyć płynność działania i ogólną „przyjazność" aplikacji.

• Walidacja może być przeprowadzana w różnych punktach i warstwach

aplikacji, a także na różnych etapach obsługi procesu biznesowego.
Często korzystne jest przeprowadzanie walidacji wielostopniowej (np. najpierw w obrębie warstwy prezentacji lub logiki biznesowej, potem – niezależnie – na poziomie bazy danych). Takie podejście, oczywiście, spowalnia aplikację, ale umożliwia zachowanie integralności danych, która jest bezwzględnie wymagana w bazach transakcyjnych.

• Realizacja walidacji na poziomie bazy danych pozwala nie tylko na zabezpieczenie przed błędami końcowego użytkownika systemu, ale także przed ewentualnymi przeoczeniami programistów, którzy implementują warstwę logiki aplikacyjnej.

• Duża część mechanizmów sprawdzających poprawność danych jest uruchamiana automatycznie przez system DBMS (np. dotyczy to deklaratywnych więzów integralności, takich jak: PK, FK, CHECK).

• Zdefiniowana przez programistę realizacja walidacji obejmuje przede wszystkim dodatkowy kod T-SQL w procedurach dodawania danych.

• Jeżeli algorytm sprawdzający poprawność określonego typu danych jest wielokrotnie wykorzystywany w bazie danych (np. sprawdzanie formatu imienia, nazwiska, adresu e-mail), można go zaimplementować w odrębnej procedurze lub funkcji użytkownika (np. `Fun_Sprawdz_Email()`), która pozwoli na redukcję nadmiarowości kodu i ujednolicenie logiki aplikacji.

• Oprogramowując we własnym zakresie logikę reagowania na nieprawidłowe dane wejściowe, zyskujemy większą kontrolę nad przebiegiem działania aplikacji, m.in. dzięki możliwości zdefiniowania własnych komunikatów o błędach i ich obsługi w wyższych warstwach aplikacji.

• W języku T-SQL można zdefiniować własny, serwerowy komunikat o błędzie za pomocą procedury systemowej `sp_addmessage`, która zapisuje w tabeli systemowej `sysmessages` (baza `[master]`) kod błędu (powyżej 50 000) i jego charakterystykę (np. komunikat, który ma być wyświetlony).

• Zgłoszenie błędu w trakcie wykonania skryptu T-SQL odbywa się za pomocą polecenia `RAISERROR`. W kodzie aplikacji (np. w języku C++) można przechwytywać zgłaszane wyjątki i odpowiednio na nie reagować.

• Własna, spójna obsługa błędów jest także korzystna ze względów bezpieczeństwa, ponieważ komunikaty zwracane bezpośrednio z serwera mogą dostarczać hakerom informacji, które pomagają we włamaniu.

Walidacja danych – przykład

```
-- Wstawianie wiersza do tabeli [Pracownik] wraz walidacją wartości parametrów.
-- Tutaj przytoczone są tylko fragmenty procedury (reszta w oddzielnym pliku).
CREATE PROC Pracownik_wstawianie_walid
/* Inne parametry. */
@Par_Nazwisko char(40), -- Parametr przekazujący nazwisko do polecenia INSERT.
/* Inne parametry. */
BEGIN
-- Obcięcie spacji wypełniających - z przodu i z tyłu.
SET @Par_Nazwisko = LTRIM(RTRIM(@Par_Nazwisko))
-- Automatyczna zamiana pierwszych liter nazwiska na kapitaliki,
-- a kolejnych znaków – na małe litery.
SET @Par_Nazwisko = UPPER(SUBSTRING(@Par_Nazwisko,1,1))
+ LOWER(SUBSTRING(@Par_Nazwisko,2,LEN(@Par_Nazwisko)))
-- Wstawianie wiersza do tabeli [Pracownik] wraz walidacją wartości parametrów.
/* Wcześniejsza część procedury. */
-- Sprawdzenie, czy nazwisko nie zawiera cyfr.
SET @Var_Licznik = 1
WHILE @Var_Licznik < LEN(@Par_Nazwisko)
BEGIN
IF (ISNUMERIC(SUBSTRING(@Par_Nazwisko,@Var_Licznik,1)) = 1)
BEGIN
RAISERROR('Nazwisko nie może zawierać cyfr.',16,1)
RETURN -1 -- Przerwanie wykonania procedury z kodem błędu.
END
SET @Var_Licznik = @Var_Licznik + 1
```

```
END
/* Dalsze polecenia procedury wstawiającej. */
```

5. Dynamiczny kod T-SQL

Dynamiczny kod T-SQL to taki ciąg poleceń, którego postać wykonywalna nie jest gotowa w momencie implementowania, lecz dopiero bezpośrednio przed jego uruchomieniem w aplikacji.

• Zastosowania:

– zapytania wielokryterialne w licznymi elementami opcjonalnymi

(np. klient sklepu internetowego może wyszukiwać produkty po rodzaju,

nazwie, typie, cenie, dostępności itp. – w momencie programowania

zapytania nie wiemy z góry, które kryteria wybierze);

– operacje wykonywane wielokrotnie z regularnymi modyfikacjami

(np. utworzenie 50 baz danych o nazwach `[B01]` – `[B50]`);

– przezwyciężanie ograniczeń standardowej składni języka T-SQL

(np. normalnie w poleceniu `SELECT TOP N` nie można podać `N`

jako zmiennej, tylko jako stałą).

Cechy kodu dynamicznego

• Zalety:

– elastyczność – możemy wygodnie dopasowywać finalną postać polecenia

T-SQL w zależności od parametrów zadanych przez użytkownika;

– mała długość – kod dynamiczny eliminuje nadmiarowość kodu

przy powtarzających się operacjach;

– przenośność – kod dynamiczny może być łatwiejszy do wykorzystania

w innych bazach i aplikacjach.

• Wady:

– obniżona wydajność – kod dynamiczny nie może być optymalizowany

na równi z kodem statycznym; ponieważ zazwyczaj istnieje wiele możliwych

wersji postaci końcowej polecenia T-SQL, jest bardzo mało prawdopodobne, że optymalizator zapytań systemu DBMS będzie w stanie ponownie wykorzystać plan wykonania (Execution Plan) dla tego polecenia; z tego względu, szczególnie przy często powtarzanych zapytaniach, kod dynamiczny może być mniej wydajny od kodu statycznego.

Metody implementacji

• Tworzenie kodu dynamicznego zazwyczaj polega na odpowiednim sklejaniu (konkatenacji) poleceń języka T-SQL w formie ciągu znaków. Sposób łączenia komend jest zależny od przebiegu przetwarzania w aplikacji (np. od opcji, które zaznacza użytkownik).

• Sklejanie ciągu T-SQL może być wykonywane:

– w obrębie obiektów kodu T-SQL przechowywanych na serwerze (np. procedur przechowywanych) – to rozwiązanie jest korzystne pod względem wydajności i bezpieczeństwa, ale niekiedy niewygodne od strony programistycznej;

– w kodzie języków wysokopoziomowych (poza bazą danych) – ciąg poleceń T-SQL jest przygotowywany np. za pomocą konstrukcji języka C++ lub Visual Basic; jest to zazwyczaj bardzo wygodne z punktu widzenia programisty, ale mniej wydajne i bezpieczne.

• Z poziomu T-SQL dowolny ciąg instrukcji może być wykonany za pomocą polecenia `sp_executesql` oraz `EXECUTE`.

• W innych językach (np. C++) przygotowany ciąg można uruchomić tak samo, jak zwykłe, tekstowe polecenie SQL, wysyłane do serwera bazodanowego.

Kod dynamiczny – przykład

```
-- Procedura wyszukująca N studentów z najwyższą średnią ocen.
-- Kod dynamiczny (wyróżniony na czerwono) pozwala na przezwyciężenie ograniczeń
-- polecenia SELECT TOP N, w którym standardowo nie można podać liczby N
-- pierwszych wierszy jako zmienną (tylko jako stałą).
```

```
CREATE PROC Najlepsi_Studenci
@Liczba_stud smallint, -- Liczba studentów, którzy mają być wyszukani.
@Rok_od smallint, -- Parametry oznaczające rok akademicki (od - do).
@Rok_do smallint
AS
BEGIN
DECLARE @Var_SQLQuery varchar(2000) -- Zmienna przechowująca zapytanie
SQL.
SET @Var_SQLQuery = ' ' -- Wyzerowanie ciągu zapytania.
-- Parametry są doklejane do ciągu instrukcji po skonwertowaniu do typu "varchar".
SET @Var_SQLQuery = 'SELECT TOP ' + CONVERT(varchar(4), @Liczba_stud)
+ ' WITH TIES Student.NrIndeksu, Student.Imie, Student.Nazwisko,
ROUND(AVG(Zaliczenie.Ocena),2) AS [Srednia] FROM Student INNER JOIN Zapisy
ON Student.NrIndeksu = Zapisy.NrIndeksu INNER JOIN ProwadzacyPrzedmiot
ON Zapisy.IdPrzydzialu = ProwadzacyPrzedmiot.IdPrzydzialu
INNER JOIN Zaliczenie ON Zapisy.IdZapisu = Zaliczenie.IdZapisu
WHERE (ProwadzacyPrzedmiot.RokPoczatek >= ' + CONVERT(varchar(4),
@Rok_od) + ' ) ' + 'AND (ProwadzacyPrzedmiot.RokKoniec <= ' + CONVER
(varchar(4), @Rok_do) + ' ) ' + 'GROUP BY Student.NrIndeksu, Student.Nazwisko,
Student.Imie, ProwadzacyPrzedmiot.RokPoczatek,
ProwadzacyPrzedmiot.RokKoniec ORDER BY [Srednia] DESC'
EXECUTE (@Var_SQLQuery) -- Wykonanie ciągu instrukcji T-SQL.
RETURN 0
END -- Koniec procedury [Najlepsi_studenci].
```

6.5 Proceduralne więzy integralności danych

Procedura wyzwalana (ang. trigger, inaczej wyzwalacz) – specjalny rodzaj procedury przechowywanej na serwerze, która jest przyporządkowana do określonej tabeli i automatycznie uruchamiana podczas modyfikacji danych w tej tabeli.

Cechy i działanie procedur wyzwalanych:

– są tworzone na poziomie tabeli i kojarzone z jedną lub większą liczbą akcji modyfikujących dane (INSERT, UPDATE, DELETE);

– jeśli wykonywana jest jedna z operacji, dla których procedura wyzwalana została zdefiniowana, jest ona automatycznie uruchamiana;

– za ich pomocą można utrzymywać złożone zasady integralności danych, których nie można zapewnić za pomocą metod deklaratywnych (np. ograniczeń CHECK, reguł).

Procedury wyzwalane typowe operacje

Za pomocą wyzwalaczy można między innymi:

• porównywać określone dane przed i po wykonaniu danej operacji;

• wykrywać i wycofywać błędne modyfikacje danych;

• automatycznie odczytywać i modyfikować dane z innych tabel, należących do tej samej lub innej bazy (lokalnej lub zdalnej);

• wywoływać procedury przechowywane (w bazie lokalnej i zdalnej).

Procedury wyzwalane zastosowania

• Zarządzanie danymi powielonymi albo obliczonymi na podstawie innych danych. Na przykład w bazie mamy tabele `[Student]` i `[Pracownik]`. Jeśli student może być jednocześnie pracownikiem, możemy wykorzystać wyzwalacze, aby po każdej modyfikacji danych osobowych w jednej tabeli sprawdzać, czy w drugiej tabeli też dane te zostały odpowiednio zmienione (jeśli istnieje tam taka sama osoba).

• Złożone więzy integralności, których nie można zapewnić metodami deklaratywnymi. Na przykład liczba studentów, należących do jednej grupy, nie może być większa od 30.

• Kaskadowa integralność referencyjna – jeśli modyfikujemy albo kasujemy wiersze z jednej tabeli, odpowiednio zmieniane lub usuwane są kaskadowo wiersze z tabeli powiązanej. W niektórych systemach bazodanowych (np. MS SQL Server 7.0) mechanizm te nie jest standardowo realizowany i trzeba wykorzystać wyzwalacze.

• Skomplikowane wartości domyślne pól w tabelach – na przykład domyślną wartością przedmiotu, który prowadzi dany wykładowca jest ten przedmiot, który najczęściej prowadził on do tej pory.

• Integralność referencyjna pomiędzy różnymi bazami danych – zmiany w jednej bazie danych automatycznie powodują zmiany w innej bazie (lokalnej lub zdalnej).

Procedury wyzwalane – rodzaje

Są dwa rodzaje procedur wyzwalanych;

• AFTER:

– deklarowane za pomocą słów AFTER lub FOR;

– są uruchamiane po wystąpieniu akcji wyzwalającej (`INSERT`, `UPDATE` lub `DELETE`) i po przetworzeniu ograniczeń tabeli;

– korzystamy z nich, jeśli zachodzi potrzeba kontrolowania modyfikacji danych, które w większości przypadków są akceptowane (rzadko wymagają wycofywania);

• INSTEAD OF:

– są uruchamiane w miejsce akcji wyzwalającej i przed przetworzeniem ograniczeń tabeli (np. takich, jak CHECK);

– są wydajniejsze od wyzwalaczy typu AFTER, jeśli kontrolują zmiany danych, które są często odrzucane, albo mają być zastępowane innymi operacjami.

Tabele INSERTED i DELETED

• Przy przetwarzaniu wyzwalaczy wykorzystywane są dwie specjalne tabele (tworzone i zarządzane przez sam DBMS), których schemat jest taki sam, jak tabeli, dla której jest zdefiniowany wyzwalacz (tabeli docelowej):

– tabela INSERTED – przechowuje kopie wierszy zmienionych przez operacje INSERT lub UPDATE;

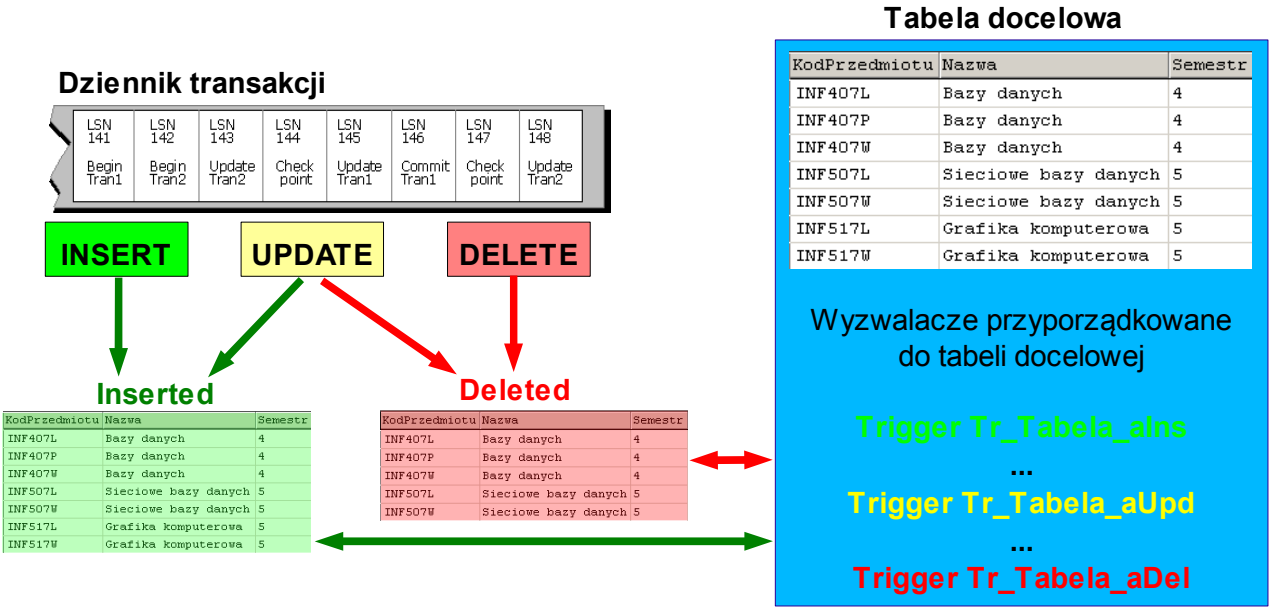
– tabela DELETED – przechowuje kopie wierszy zmienionych przez operacje DELETE lub UPDATE (tabela UPDATED nie jest potrzebna, gdyż operacja UPDATE to odpowiednik ciągu dwóch operacji: DELETE i INSERT).

• Zanim operacja zmiany danych zostanie zatwierdzona, system sprawdza, czy do danej operacji nie jest przypisana procedura wyzwalana i dopiero po jej ewentualnym uruchomieniu, transakcja jest zatwierdzana, a kopie wierszy w INSERTED i DELETED są kasowane.

• Obsługa tabel DELETED i INSERTED zależy od rodzaju wyzwalaczy:

– w przypadku wyzwalaczy typu AFTER tabele te są widokami na tabelę docelową (tabelami wirtualnymi), zawierającymi zmodyfikowane wiersze;

– w przypadku wyzwalaczy typu INSTEAD OF są to fizyczne tabele tymczasowe, rzeczywiście odrębne w stosunku do tabeli docelowej.



Rys. 6.4. Działanie procedur wyzwalanych.

Źródło rysunku dziennika transakcji: MICROSOFT, Books On-Line, Microsoft Corp. 1988 –2000.
Zrzuty ekranowe tabel pochodzą z programu MS „Query Analyzer".

Działanie wyzwalacza typu AFTER

1. W tabeli docelowej zgłaszana jest operacja `INSERT`, `UPDATE` lub `DELETE`.
2. System sprawdza, czy dla tej operacji nie ma zdefiniowanego wyzwalacza.
3. Jeśli nie ma wyzwalacza, przetwarzane są ograniczenia tabeli (np. CHECK) i jeżeli są one spełnione, transakcja jest zatwierdzana.
4. Jeśli jest zdefiniowany wyzwalacz, automatycznie tworzone są tabele `INSERTED` i `DELETED` z kopiami zmodyfikowanych wierszy. Są to tabele wirtualne – widoki, zawierające zmienione wiersze.
5. Przetwarzane są ograniczenia i dokonywane są odpowiednie modyfikacje danych w tabeli docelowej, ale nie są one zatwierdzane (`COMMIT`).
6. Uruchamiany jest wyzwalacz typu AFTER, który może jawnie wycofać operację (`ROLLBACK`), albo ją niejawnie zatwierdzić (jeśli nic nie zrobi).
7. Tabele `INSERTED` i `DELETED` są automatycznie usuwane.

Działanie wyzwalacza INSTEAD OF

1. W tabeli docelowej zgłaszana jest operacja `INSERT`, `UPDATE` lub `DELETE`.
2. System sprawdza, czy dla tej operacji nie ma zdefiniowanego wyzwalacza.
3. Jeśli nie ma wyzwalacza, przetwarzane są ograniczenia tabeli (np. `CHECK`) i jeżeli są one spełnione, transakcja jest zatwierdzana.
4. Jeśli jest zdefiniowany wyzwalacz, automatycznie tworzone są tabele `INSERTED` i `DELETED` z kopiami zmodyfikowanych wierszy. Są to tabele tymczasowe – fizyczne, odrębne w stosunku do tabeli docelowej.
5. Uruchamiany jest wyzwalacz typu INSTEAD OF, który wykonuje określone operacje zamiast tych, które go wywołały. Jeśli na przykład jest on pusty, to w tabeli docelowej nie będą dokonane żadne zmiany.
6. W kontekście powyższych operacji przetwarzane są ograniczenia i dokonywane są odpowiednie modyfikacje danych w tabeli docelowej.
7. Tabele `INSERTED` i `DELETED` są automatycznie usuwane.

2. Implementacja i zarządzanie wyzwalaczami.

Projektowanie wyzwalaczy

Aby zastosować wyzwalacze, jako metodę zapewnienia reguł integralności w bazie danych, należy wykonać poniższe kroki.

1. Zidentyfikować obszary bazy i procesy, które wymagają skomplikowanych więzów integralności.

2. Sprawdzić, czy powyższych reguł nie można zaimplementować w postaci ograniczeń deklaratywnych (np. CHECK, FOREIGN KEY, UNIQUE).

3. Utworzyć odpowiednie wyzwalacze typu AFTER lub INSTEAD OF.

4. W razie potrzeby ustalić kolejność wykonywania wyzwalaczy w ramach danej tabeli.

T-SQL: Tworzenie procedury wyzwalanej – składnia

```
CREATE TRIGGER trigger_name
ON { table | view }
[ WITH ENCRYPTION ]
{
{ { FOR | AFTER | INSTEAD OF } { [ INSERT ] [ , ] [ UPDATE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS
[ { IF UPDATE ( column )
[ { AND | OR } UPDATE ( column ) ]
[ ...n ]
| IF ( COLUMNS_UPDATED ( ) { bitwise_operator } updated_bitmask )
{ comparison_operator } column_bitmask [ ...n ]
} ]
sql_statement [ ...n ]
}
}
```

Polecenia, których nie można używać w wyzwalaczach

- ALTER DATABASE,
- CREATE DATABASE,
- DISK INIT,
- DISK RESIZE,
- DROP DATABASE,
- LOAD DATABASE,
- LOAD LOG,
- RECONFIGURE,
- RESTORE DATABASE,
- RESTORE LOG.

T-SQL: Tworzenie wyzwalacza typu AFTER – przykład

```
-- Wyzwalacz typu AFTER: nie pozwala na usunięcie studenta: przy próbie
-- wykonania tej operacji wyświetla komunikat o błędzie i wycofuje operację.
CREATE TRIGGER Tr_Student_aDel ON Student
AFTER DELETE
AS
RAISERROR ('Nie można usuwać studentów.', 16, 1)
ROLLBACK TRAN -- Cofnięcie transakcji – zmiany danych.
GO
```

T-SQL: Tworzenie wyzwalacza typu INSTEAD OF – przykład

```
-- Prosty wyzwalacz typu INSTEAD OF. Zamiast aktualizacji lub usuwania wierszy
-- w tabeli [Zaliczenie] wyświetlany jest komunikat o błędzie, a tabela pozostaje
-- nie zmieniona.
CREATE TRIGGER Tr_Zaliczenie_iUpd_Del ON Zaliczenie
INSTEAD OF UPDATE, DELETE
AS
RAISERROR ('Nie można modyfikować zaliczeń.', 16, 1)
GO
```

Zarządzanie wyzwalaczami

• Wyzwalacze są procedurami przyporządkowanymi do konkretnych tabel. Nie są one zatem widoczne razem ze zwykłymi procedurami przechowanymi.

• W programie „Enterprise Manager" można tworzyć, modyfikować i usuwać wyzwalacze w funkcji „All tasks – Manage Triggers" (menu kontekstowe na wybranej tabeli).

• W programie „Query Analyzer" wyzwalacze są widoczne w węźle „User Tables –<Table> – Triggers", skąd można je pobierać do edycji („Edit").

• Z poziomu kodu T-SQL do wyświetlenia informacji o wyzwalaczach w danej tabeli służy procedura systemowa `sp_helptrigger <nazwa_tabeli>`.

• Bieżący kod T-SQL wyzwalacza możemy sprawdzić za pomocą procedury `sp_helptext <nazwa_wyzwalacza>`.

• Jeżeli w tabeli jest kilka wyzwalaczy dla jednej operacji, za pomocą procedury `sp_settriggerorder` można ustalić, który z nich będzie wykonywany jako pierwszy albo ostatni. W przeciwnym razie kolejność wykonywania nie jest deterministyczna.

Wyzwalacze a procedury przechowywane

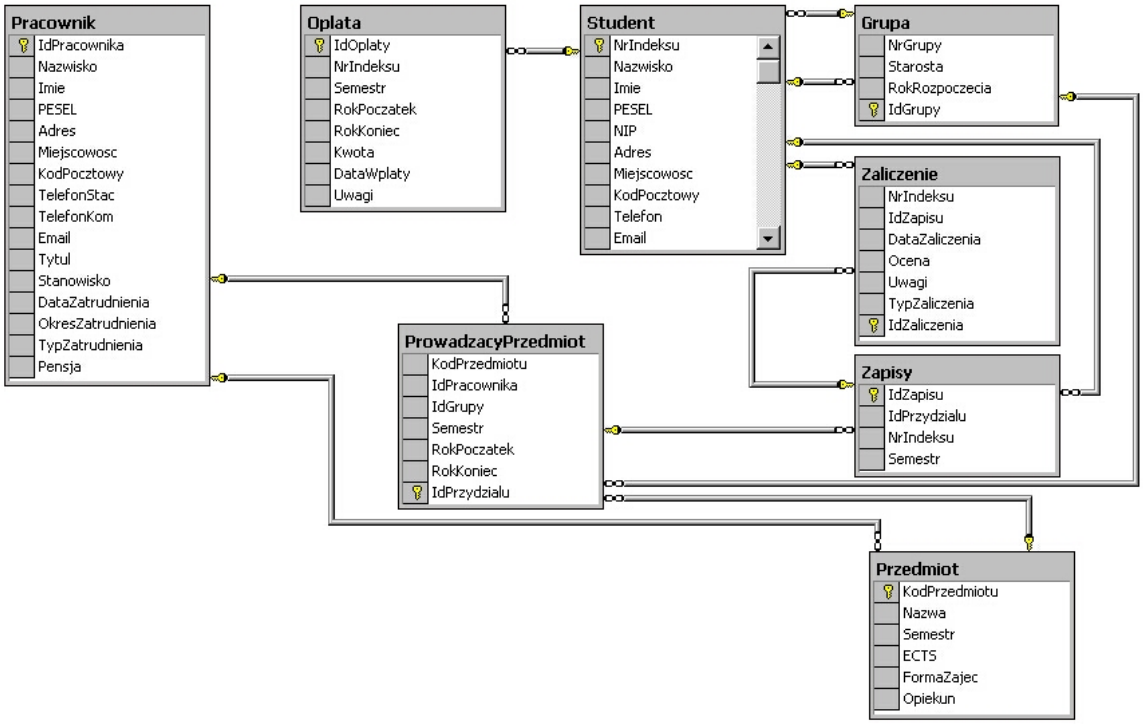
Zaawansowane reguły integralności można realizować alternatywnie za pomocą zwykłych procedur przechowywanych. Są jednak istotne różnice pomiędzy oboma rozwiązaniami.

• Jeśli wyzwalacz jest aktywny, kontroluje on wszystkie zmiany (`INSERT`, `UPDATE`, `DELETE`) dokonywane w tabeli, niezależnie od miejsca, z którego zostały one wywołane (wyjątkiem są polecenia `TRUNCATE TABLE` i `BULK INSERT`, które standardowo nie uruchamiają wyzwalaczy). Wyzwalacz pozwala zatem na wygodne zdefiniowanie w pojedynczym miejscu więzów integralności, które będą zawsze przestrzegane, nawet, jeśli zmiany pochodzą z różnych poleceń, procedur, aplikacji zewnętrznych itp.

• Jeśli reguła integralności jest implementowana poza wyzwalaczem, na przykład w kodzie procedury przechowywanej, jest ona spełniana tylko lokalnie – w obrębie samej procedury. A zatem, przy rezygnacji z wyzwalaczy programista musi pamiętać o dodaniu odpowiedniego kodu, zapewniającego więzy integralności, we wszystkich miejscach, w których polecenia T-SQL dokonują zmian w tabeli.

• Ponieważ wyzwalacze są uruchamiane automatycznie, ich działanie jest trudniejsze do kontrolowania od zwykłych procedur (np. wyzwalacze mogą być zagnieżdżane i uruchamiane kaskadowo lub rekurencyjnie).

• Wyzwalacze są zazwyczaj mniej wydajne od procedur (m.in. ze względu na automatyczne tworzenie odrębnych struktur danych, np. tabel `[INSERTED]` i `[DELETED]`). Z tego względu przy dużej intensywności zmian w tabeli, wyzwalacze mogą obniżać wydajność całego systemu.



Rys. 6.5. Schemat bazy danych „Uczelnia”.

3. Studium przypadku – wyższa uczelnia techniczna.

Analiza skomplikowanych reguł integralności w bazie „Uczelnia”

• Liczba studentów zapisanych do danej grupy nie może przekraczać 30 osób.

• Student może być starostą tylko tej grupy, do której sam należy.

• Jeśli student jest zapisany w bazie, nie można go skasować, a jedynie wypisać, wypełniając odpowiednio pole `[DataWypisania]` zamiast wartości domyślnej (chyba, że student był już wypisany wcześniej).

• Nie można zapisać na przedmiot studenta, który jest wypisany.

• W danym semestrze student może mieć maksymalnie 3 oceny z przedmiotu egzaminacyjnego i 2 z nieegzaminacyjnego. Tylko jedna, najpóźniejsza ocena może być pozytywna. Student może uzyskać maksymalnie jedną ocenę z danego przedmiotu dziennie.

Rozwiązanie (1)

• Reguła integralności: „Liczba studentów zapisanych do danej grupy nie może przekraczać 30 osób.”

• Opis rozwiązania:

– Wyzwalacz typu AFTER na tabeli `[Student]`, zdefiniowany dla operacji `INSERT` i `UPDATE`.

– Po każdej zmianie danych w tabeli, wyzwalacz:

1) z tabeli `[INSERTED]` pobiera do zmiennej `@Var_IdGrupy` wartość pola `[IdGrupy]` (klucza obcego, wskazującego na wiersz w tabeli `[Grupa]`);

2) w tabeli [`Student]` zlicza wszystkie wiersze, w których wartość pola `[IdGrupy]` jest równa zmiennej `@Var_IdGrupy`; liczbę wierszy zapisuje w zmiennej `@Var_LiczbaStud`;

3) jeśli wartość zmiennej `@Var_LiczbaStud` jest większa od 30, wówczas operacja jest wycofywana przez polecenie `ROLLBACK`.

Rozwiązanie (2)

• Reguła integralności:

„Student może być starostą tylko tej grupy, do której sam należy.”

• Opis rozwiązania:

– Wyzwalacz typu AFTER na tabeli `[Grupa]`, zdefiniowany

dla operacji `INSERT` i `UPDATE`.

– Po każdej zmianie danych w tabeli, wyzwalacz:

1) z tabeli `[INSERTED]` pobiera do zmiennych `@Var_IdGrupy`, `@Var_Starosta`

odpowiednio wartości pól `[IdGrupy]` oraz `[Starosta]` (klucza obcego,

wskazującego na wiersz w tabeli `[Student]`);

2) z tabeli `[Student]` pobiera do zmiennej `@Var_GrupaStud` wartość pola

`[IdGrupy]` dla studenta, którego wartość pola `[NrIndeksu] = @Var_Starosta`;

3) jeśli zmienne się różnią, czyli `@Var_IdGrupy <> @Var_GrupaStud`,

to operacja jest wycofywana poleceniem `ROLLBACK`.

Rozwiązanie (3)

• Reguła integralności:

„Jeśli student jest zapisany w bazie, nie można go skasować, a jedynie

wypisać, wypełniając odpowiednio pole `[DataWypisania]` zamiast wartości

domyślnej (chyba, że student był już wypisany wcześniej).”

• Opis rozwiązania:

– Wyzwalacz typu INSTEAD OF na tabeli `[Student]`,

zdefiniowany dla operacji `DELETE`.

– Po każdej zmianie danych w tabeli, wyzwalacz:

1) z tabeli `[INSERTED]` pobiera do zmiennych `@Var_NrIndeksu`,

`@Var_DataWypisania` odpowiednio wartości pól `[NrIndeksu]`,

`[DataWypisania]`;

2) jeśli `@Var_DataWypisania = '1900-01-01'` (student nie jest wypisany),

to zamiast kasowania jest wykonywana aktualizacja pola: `[DataWypisania]` nową wartością z funkcji `GETDATE ( )`.

Rozwiązanie (4)

• Reguła integralności:

„Nie można zapisać na przedmiot studenta, który jest wypisany.”

• Opis rozwiązania:

– Wyzwalacz typu AFTER na tabeli `[Zapisy]`, zdefiniowany dla operacji `INSERT`, `UPDATE`.

– Po każdej zmianie danych w tabeli, wyzwalacz:

1) z tabeli `[INSERTED]` pobiera do zmiennej `@Var_NrIndeksu` wartość pola `[NrIndeksu];`

2) z tabeli `[Student]` pobiera do zmiennej `@Var_DataWypisania` wartość pola `[DataWypisania];`

3) jeśli `@Var_DataWypisania <> '1900-01-01'` (wartość nie jest domyślna, czyli student jest wypisany), to operacja jest wycofywana poleceniem `ROLLBACK`.

Rozwiązanie (5)

• Reguła integralności:

„W danym semestrze student może mieć maksymalnie 3 oceny z przedmiotu egzaminacyjnego i 2 z nieegzaminacyjnego. Tylko jedna, najpóźniejsza ocena może być pozytywna. Student może uzyskać maksymalnie jedną ocenę z danego przedmiotu dziennie.”

• Opis rozwiązania:

– Wyzwalacz typu AFTER na tabeli `[Zaliczenie]`, zdefiniowany dla operacji `INSERT` i `UPDATE`.

– Po każdej zmianie danych w tabeli, wyzwalacz:

1) z tabeli `[INSERTED]` pobiera do zmiennych `@Var_IdZaliczenia`, `@Var_IdZapisu`, `@Var_Ocena`, `@Var_TypZaliczenia`, `@Var_DataZaliczenia`, `@Var_Termin` odpowiednio wartości pól `[IdZaliczenia]`, `[IdZapisu]`, `[Ocena]`, `[TypZaliczenia]`, `[DataZaliczenia]`, `[Termin]`;

2) tworzy tabelę tymczasową `[#Tmp_Zaliczenie]` i wypełnia ją wszystkimi zaliczeniami, które student ma z bieżącego przedmiotu (`@Var_IdZapisu`);

3) wycofuje operację (`ROLLBACK`), jeżeli w tabeli `[#Tmp_Zaliczenie]` istnieje więcej niż 3 wiersze;

4) wycofuje operację (`ROLLBACK`), jeżeli `@Var_TypZaliczenia = 'zaliczenie'` i w tabeli `[#Tmp_Zaliczenie]` istnieje więcej niż 2 wiersze;

5) wycofuje operację (`ROLLBACK`), jeżeli w tabeli `[#Tmp_Zaliczenie]` istnieje więcej niż 1 wiersz, gdzie wartość pola `[Ocena] > 2,0`;

6) wycofuje operację (`ROLLBACK`), jeżeli `@Var_Ocena > 2,0` i w tabeli `[#Tmp_Zaliczenie]` jest wiersz, gdzie `[IdZaliczenia] <> @Var_IdZaliczenia AND [DataZaliczenia] > @Var_DataZaliczenia`;

7) wycofuje operację (`ROLLBACK`), jeżeli `@Var_Ocena = 2.0` i w tabeli `[#Tmp_Zaliczenie]` jest wiersz, gdzie [`Ocena] > 2.0 AND [IdZaliczenia] <> @Var_IdZaliczenia AND [DataZaliczenia] < @Var_DataZaliczenia`;

8) wycofuje operację (`ROLLBACK`), jeżeli w tabeli `[#Tmp_Zaliczenie]` jest inny wiersz, gdzie `[DataZaliczenia] = @Var_DataZaliczenia`;

9) sprawdza i automatycznie nadaje numer kolejnego zaliczenia (pole `[Termin]`):

- jeśli jest to jedyne zaliczenie studenta z tego przedmiotu, musi to być termin "1";
- jeśli łącznie z nowo wprowadzanym są 2 zaliczenia w tabeli, to o terminach "1", "2", "3" decyduje ich data;

10) jawnie usuwa tabelę `[#Tmp_Zaliczenie]`.

Kod zaimplementowanych wyzwalaczy jest umieszczony w oddzielnym pliku z przykładami.

Pytania kontrolne[1]

1. Od czego zależy kolejność wstawiania i usuwania danych do tabel w relacyjnej bazie danych?
2. Jakie korzyści możemy odnieść wykorzystując do przetwarzania danych obiektu kodu T-SQL przechowywane na serwerze?
3. Czy za pomocą polecenia SELECT można zwrócić tylko kolumny, które fizycznie istnieją w tabeli źródłowej?
4. Jaka jest różnica w działaniu klauzul ORDER BY oraz GROUP BY?
5. Jaki rodzaj złączenia jest najczęściej stosowany praktycznie do tworzenia zestawień, raportów i statystyk?
6. W jakiej sytuacji nie można zastąpić podzapytania żadną inną konstrukcją (nie licząc dwustopniowego wykonywania zapytania z wykorzystaniem tabel tymczasowych)?
7. Na czym polega podstawowa przewaga procedury wyzwalanej nad procedurą przechowywaną, która realizuje tę samą operację?
8. Co to jest kod dynamiczny i do czego jest wykorzystywany?

[1] Odpowiedzi na pytania można znaleźć w Dodatku A.

Zadania egzaminacyjne

Poniżej zamieszczone są przykładowe pytania, których forma i zakres merytoryczny są zbliżone do pytań egzaminacyjnych z przedmiotu *Bazy danych* (INF407).

Pytanie 6_1 (dopasowanie odpowiedzi)

Zadaniem procedury przechowywanej [Wycieczki_Wstaw] jest wstawienie nowego wiersza do tabeli [Wycieczki], zgodnie z danymi przekazanymi przez parametry. Ustal prawidłową kolejność linii kodu T-SQL procedury (MS SQL Server 2005/2008).

Tabela [Wycieczki].

NrWycieczki	Miejsce	DataWyjazdu	DataPowrotu	PESEL	ImięUczestnika	NazwiskoUczestnika
15/2004	Praga	2004-02-13	2004-02-15	72120512432	Andrzej	Nowak
15/2004	Praga	2004-02-13	2004-02-15	65050678098	Katarzyna	Kowalska
15/2004	Praga	2004-02-13	2004-02-15	81103098023	Anna	Górska
16/2004	Włochy	2004-02-20	2004-02-29	53012743123	Janusz	Romanowski
16/2004	Włochy	2004-02-20	2004-02-29	49030187234	Maciej	Niedzielski
...	...		...	...	...	...

```
1. CREATE PROCEDURE Wycieczki_Wstaw
2. @Par_NrWycieczki varchar(8),
   @Par_Miejsce varchar(30),
   @Par_DataWyjazdu datetime,
   @Par_DataPowrotu datetime,
   @Par_PESEL char(11),
   @Par_ImieUczestnika varchar(30),
3. @Par_NazwiskoUczestnika varchar(40)
4. AS
5. BEGIN
6. INSERT Wycieczki
7. (NrWycieczki, Miejsce, DataWyjazdu, DataPowrotu, PESEL, ImieUczestnika,
   NazwiskoUczestnika)
8. VALUES
9. (@Par_NrWycieczki, @Par_Miejsce, @Par_DataWyjazdu, @Par_DataPowrotu,
   @Par_PESEL, @Par_ImieUczestnika, @Par_NazwiskoUczestnika)
10. END
```

Pytanie 6_2 (wielokrotny wybór, 3 odpowiedzi)

Procedury przechowywane (ang. stored procedures) (wybierz 3 najlepiej pasujące odpowiedzi):

1. **są obiektami kodu SQL przechowywanymi na serwerze.**
2. **zwiększają wydajność wykonywania często powtarzanych zapytań.**
3. **ułatwiają zabezpieczenie aplikacji przed atakami "wstrzyknięcia kodu SQL" (ang. SQL injection).**

4. zwiększają ruch w sieci.
5. służą do zapisywania kopii zapasowych bazy danych (ang. backup).
6. są tabelami wirtualnymi.
7. są przechowywane w pamięci podręcznej i zawierają dane, które są najczęściej wykorzystywane w bazie.

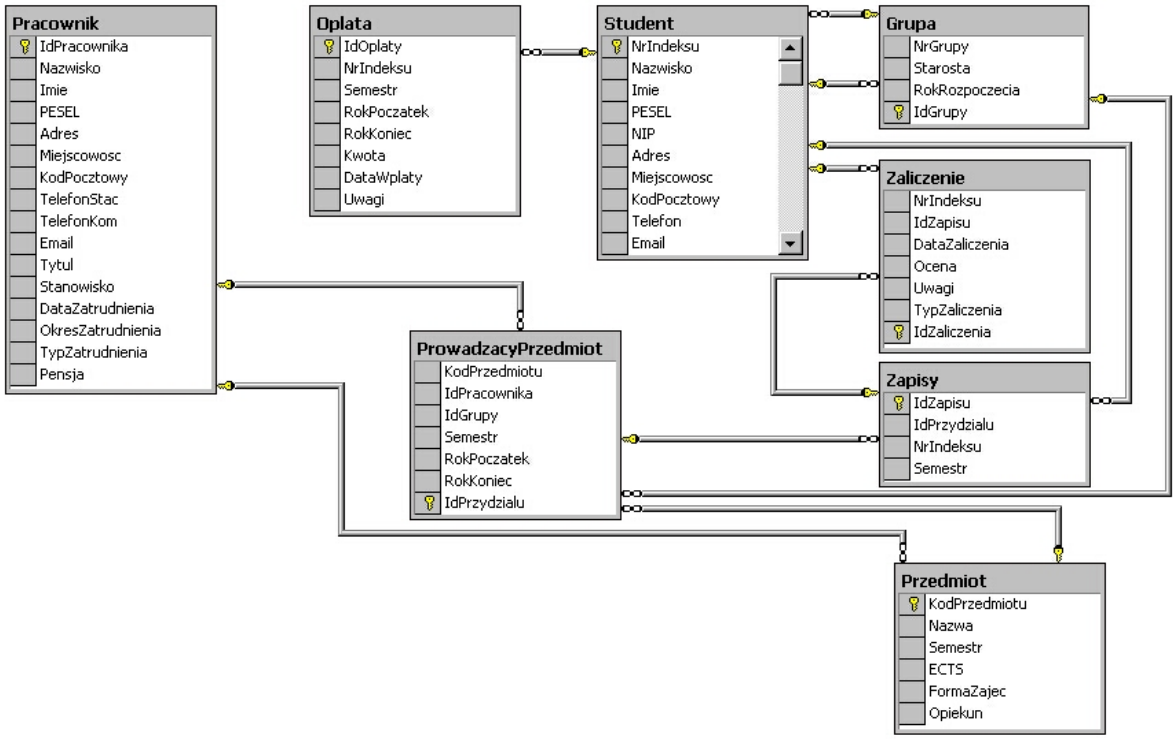
Pytanie 6_3 (wielokrotny wybór, 2 odpowiedzi)

Polecenie TRUNCATE TABLE (wybierz 2 najlepiej pasujące odpowiedzi):

1. **usuwa wszystkie wiersze z tabeli bez rejestrowania operacji w dzienniku transakcji.**
2. **ustawia licznik pola autonumerycznego (IDENTITY) na wartość początkową (SEED).**
3. usuwa wszystkie relacje wychodzące z danej tabeli.
4. usuwa bezpowrotnie strukturę tabeli wraz z danymi.
5. pozwala na wybranie wierszy do usunięcia poprzez klauzulę WHERE.
6. usuwa z tabeli tylko te wiersze, które przekraczają określony, maksymalny rozmiar tabeli.

Pytanie 6_4 (wielokrotny wybór, 3 odpowiedzi)

Schemat bazy [Uczelnia] jest przedstawiony na poniższym rysunku. Opisz, jakie operacje



wykonuje poniższa procedura (wybierz 3 najlepiej pasujące odpowiedzi).

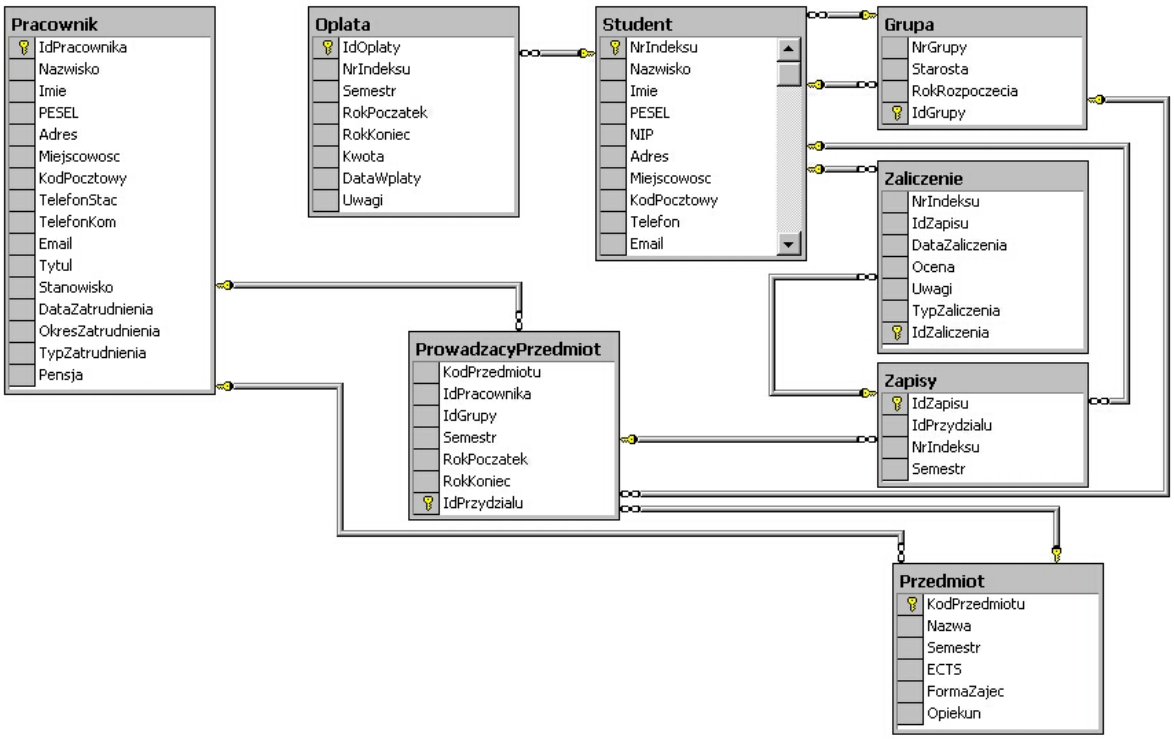
1. **Zwraca grupy, których rok rozpoczęcia studiów jest większy lub równy podanemu parametrowi.**
2. **Wyświetla wszystkie grupy studentów (nawet jeśli nie mają starostów).**
3. **Wyświetla w jednej kolumnie imię i nazwisko starosty grupy.**
4. Wyświetla tylko te grupy, które posiadają starostów.
5. Wyświetla wszystkie możliwe kombinacje grup i starostów.
6. Wyświetla tylko te grupy, których rok rozpoczęcia studiów jest większy lub równy 1990.

```
-------------------------------------
CREATE PROC Grupa_szukanie
@Par_RokRozpoczecia smallint = 1990
AS
BEGIN
SELECT NrGrupy AS [Nazwa grupy], RokRozpoczecia AS [Rok rozpoczecia], RTRIM(Imie) +
' ' + RTRIM(Nazwisko) AS [Starosta], NrIndeksu AS [Nr indeksu]
FROM Grupa
      LEFT OUTER JOIN Student
      ON Grupa.Starosta = Student.NrIndeksu
WHERE RokRozpoczecia >= @Par_RokRozpoczecia
END
GO
-------------------------------------
```

Pytanie 6_5 (prawda / fałsz)

W poniższej procedurze, wybierającej studentów z danej grupy, podzapytanie można zastąpić złączeniem.{TRUE}

```
-----------------------------------------------
CREATE PROCEDURE Student_Grupa_podzapytanie
@Par_NrGrupy varchar(7) = ''
AS
BEGIN
SELECT NrIndeksu AS [Numer indeksu], Imie, Nazwisko
FROM Student
WHERE IdGrupy IN
      (SELECT IdGrupy
       FROM Grupa
       WHERE NrGrupy LIKE '%' + @Par_NrGrupy + '%')
ORDER BY Nazwisko ASC, Imie ASC
END
GO
-----------------------------------------------
```



Pytanie 6_6 (uzupełnianie krótkimi wyrazami)

Uzupełnij poniższe polecenie SQL tak, aby w tabeli [Pracownik] modyfikowało nazwisko na "Nowak" pracownikowi, którego [IdPracownika] = 10.

[ ] `Pracownik (`**`UPDATE`**`)`

[ ] `Nazwisko = 'Nowak' (`**`SET`**`)`

[ ] `IdPracownika = 10 (`**`WHERE`**`)`