**Zaawansowane systemy baz danych**

*Projekt – etap 2*

Krzysztof Wyszyński

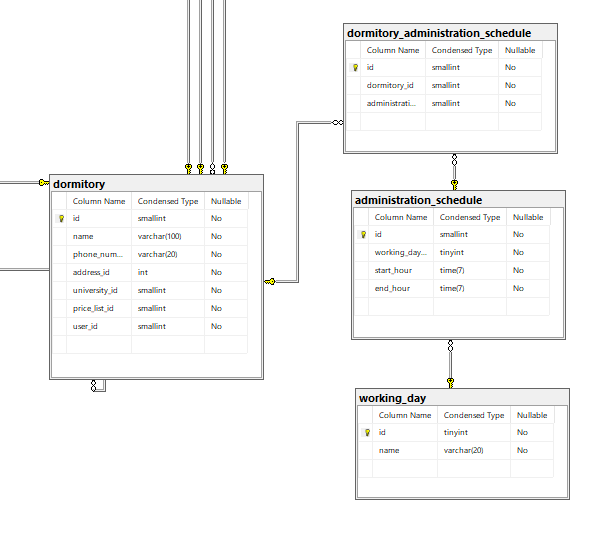
*W etapie drugim baza danych uzupełniona została większą ilością danych – w każdym Domu Studenckim znajduje się 200 pomieszczeń. W każdym pomieszczeniu znajduje się jeden student. W związku z tym łączna liczba studentów w bazie wynosi 2600.*

Podczas analizy etapu 1. wystąpiły błędy, które skorygowane zostały w etapie drugim:

1. Relacja degree/department/university – wydział może mieć wiele kierunków, a uczelnia wiele wydziałów (dwie relacje one-to-many)
2. Kolumna *number* w tabeli *room* nie jest już typu SMALLINT – typ VARCHAR(10) umożliwi wstawianie liter do numeru pokoju, np. *23A*
3. Typ CHAR(9) dla numeru telefonu nie jest dobrym rozwiązaniem – znacząco ogranicza zakres wprowadzanych wartości oraz nie uwzględnia numerów kierunkowych. Został zastąpiony przez VARCHAR(20)
4. Usunięto flagę *is\_working* z tabeli *doorkeeper*
5. Zmodyfikowano typy danych w tabeli *price\_list* – uwzględniono możliwość wprowadzania liczb zmiennoprzecinkowych

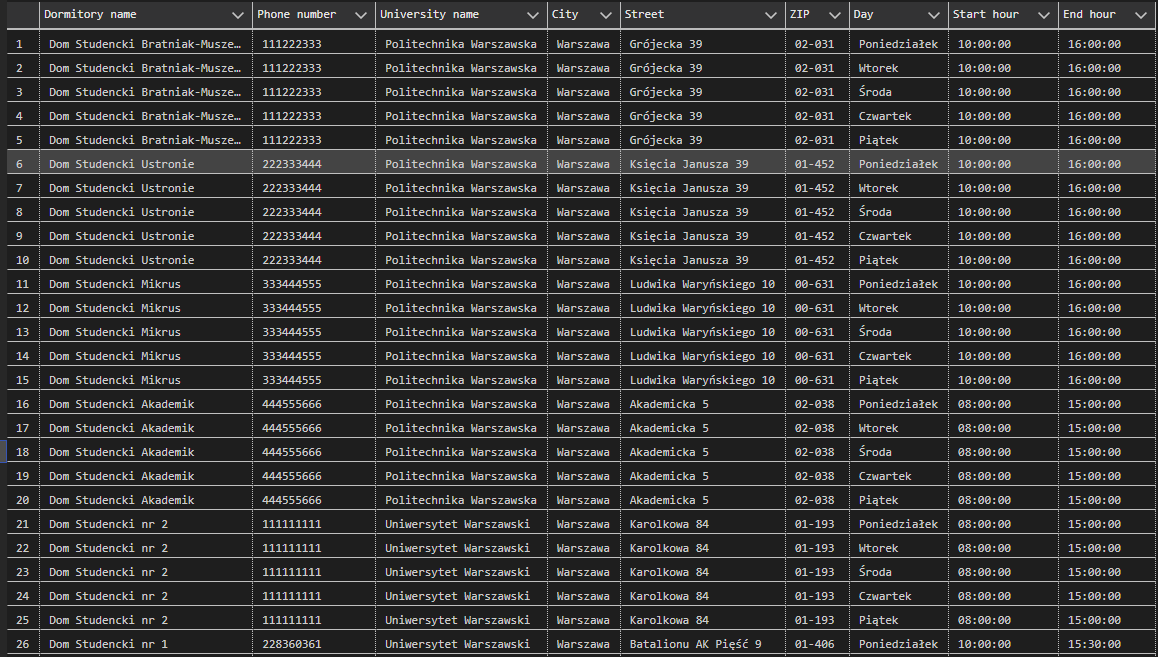
Pozostałe modyfikacje:

1. W tabeli *dormitory* znajduje się kolumna *user\_id*, dzięki której możliwe będzie zastosowanie RLS (Row Level Security) – ograniczenie dostępu do danych w tabeli dla poszczególnych użytkowników
2. Utworzono tabele zawierające godziny pracy administracji dla poszczególnych Domów Studenckich:



1. **Perspektywy**

W celu udostępnienia użytkownikom czytelnej, łatwo dostępnej informacji wygenerowane zostały następujące widoki:

1. *dormitories* – informacje o danych adresowych Domów Studenckich oraz godzinach pracy administracji w poszczególne dni i numerze kontaktowym. Może zostać wykorzystana przez dowolnego użytkownika, jako informacja powszechnie dostępna
2. *students* – informacje o studentach zamieszkujących Domy Studenckie. Może zostać wykorzystana przez pracowników administracji poszczególnych akademików

Obraz zawierający tekst, okno, sprzęt elektroniczny, komputer

Opis wygenerowany automatycznie

1. *rooms* – informacje o pojemności pokoju, jego numerze, nazwie Domu Studenckiego do którego należy oraz aktualnej liczbie lokatorów

Obraz zawierający stół

Opis wygenerowany automatycznie

1. *doorkeepers* – informacje o pracownikach portierni Domów Studenckich

Obraz zawierający tekst, czarny, zewnętrzne, tablica wyników

Opis wygenerowany automatycznie

1. *Obraz zawierający tekst, monitor, ściana, telewizja

   Opis wygenerowany automatycznieadministration\_workers* – informacje o pracownikach administracji Domów Studenckich
2. *issues* – informacje o zgłoszonych skargach w Domach Studenckich, wraz z danymi osoby zgłaszającej oraz studenta, na którego została złożona skarga

Obraz zawierający tekst, monitor, czarny

Opis wygenerowany automatycznie

1. *announcements* – informacje o ogłoszeniach administracji Domów Studenckich wraz z danymi kontaktowymi do autora ogłoszenia

Obraz zawierający tekst, czarny, tablica wyników

Opis wygenerowany automatycznie

1. **Indeksy**

Podczas zajęć wykładowych zostaliśmy zapoznani z indeksami klastrowymi oraz nieklastrowymi. Pierwsze z nich służą do logicznego i fizycznego uporządkowania danych w tabeli. Pojedyncza tabela może posiadać tylko jeden indeks klastrowy, który może zostać nałożony na jedną bądź wiele kolumn. Indeks nieklastrowy to kopia wybranych kolumn z tabeli, pozwalająca na szybkie wyszukiwanie danych z wybranych kolumn – zamiast skanowania wykonywany jest *Seek*– system bazodanowy nie przemierza całej tabeli w poszukiwaniu danych.

* 1. **Indeksy klastrowe**

System bazodanowy Microsoft SQL Server Management Studio domyślnie nakłada klastrowe indeksy na klucze główne(PK) w tabelach. Niekoniecznie jest to działanie pożądane – klucz główny jest wartością unikatową w danej tabeli, natomiast indeksy klastrowe są skuteczne w przypadku danych często powtarzających się, często sortowanych, danych z pewnego pożądanego przedziału, mniejszych lub większych od określonej wartości.

Dobrym przykładem pozwalającym na wykorzystanie indeksu klastrowego w mojej bazie danych jest tabela *person*, przechowująca dane osobowe, m.in. numer PESEL. Uważam, że kolumna ta może być potencjalnie często wykorzystywana podczas zapytań. Poniżej przedstawiono rezultat wyszukiwania wszystkich osób o numerze PESEL większym niż 96, na tabeli z domyślnie założonym indeksie klastrowym na kluczu głównym:

select \* from person where pesel > '96'

set statistics io on

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

Jak przedstawiono powyżej, system bazodanowy przeskanował całą tabelę wykorzystując domyślny indeks klastrowy założony na kluczu głównym. Odczytano 19 stron bazodanowych.

Poniżej przedstawiono rezultat tego samego zapytania, po zdjęciu indeksu klastrowego z klucza głównego oraz nałożeniu go na kolumnę PESEL:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

Jak można zauważyć, tym razem udało się uzyskać *Seek*, a ilość odczytanych stron wyniosła 11.

W przypadku tabeli *address* uznałem, że zapytania często odnosić mogą się do konkretnego miasta. Dodatkowo, jest to kolumna o danych powtarzających się, a zatem dobrym rozwiązaniem będzie założyć na nią indeks klastrowy. Tak jak w poprzednim przykładzie, poniżej przedstawiono rezultaty uzyskane w formie domyślnej:

select \* from address where city='Warszawa'

set statistics io on

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

Oto wyniki po modyfikacji indeksu klastrowego:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

Jak można zauważyć, ilość odczytów została znacząco zmniejszona.

W przypadku tabeli *issue* oraz *announcement* indeksy klastrowe nałożyłem na kolumnę *date*, z porządkiem malejącym*.* W tabeli *room* ustawiłem indeks klastrowy na kolumny *capacity* oraz *dormitory\_id*.

* 1. **Indeksy nieklastrowe**

W punkcie 2.1. dane w tabelach uporządkowane zostały fizycznie oraz logicznie. Aby dodatkowo przyspieszyć proces wyszukiwania rekordów po kolumnach, które mogą być często uwzględniane podczas wykonywania zapytań, można wykorzystać indeksy nieklastrowe. Zakładam, że w mojej bazie często następować będzie wyszukiwanie osób po imieniu oraz nazwisku. Poniżej przedstawiono rezultat przykładowego zapytania zawierającego podane kryteria:

select \* from person where first\_name='Arnold' and last\_name='Eglise'

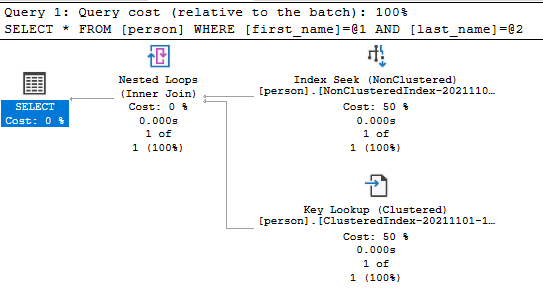
Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

Jak można zauważyć, cała tabela została przeskanowana – odczytano 20 stron. Poniżej przedstawiono rezultat tego samego zapytania po nałożeniu indeksu nieklastrowego na kolumny *first\_name* oraz *last\_name:*



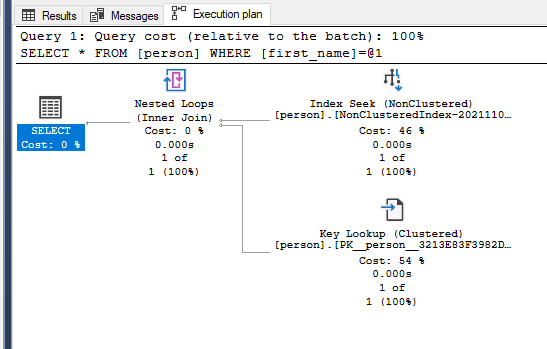
Obraz zawierający tekst

Opis wygenerowany automatycznie

Tym razem przeszukany został indeks nieklastrowy, co przyniosło bardziej wydajny rezultat.

Wygenerowany indeks to indeks kompozytowy. Porównajmy jego działanie z sytuacją, gdy stworzone zostaną dwa indeksy proste dla kolumn *first\_name* i *last\_name:*

select \* from person where first\_name='Darya'



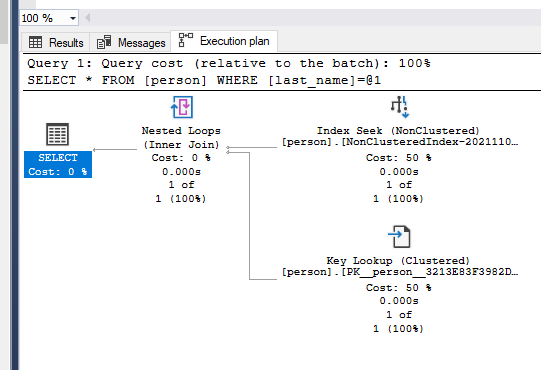
Obraz zawierający tekst

Opis wygenerowany automatycznie

select \* from person where last\_name='Mathelin'

Obraz zawierający tekst

Opis wygenerowany automatycznie



Powtórzmy eksperyment przy użyciu indeksu kompozytowego:

select \* from person where first\_name='Darya'

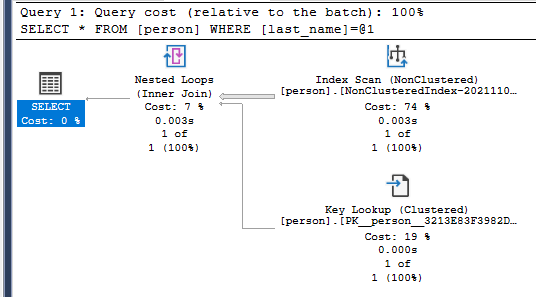
Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

select \* from person where last\_name='Mathelin'



Obraz zawierający tekst

Opis wygenerowany automatycznie

Z powyższych zrzutów ekranu można wywnioskować, że w przypadku indeksu kompozytowego nastąpiło przeskanowanie całej struktury podczas filtrowania osoby po nazwisku. Wynika to z faktu, iż w indeksie kompozytowym istotna jest kolejność kolumn, które wchodzą w jego skład. Z tego powodu filtrowanie po imieniu przyniosło dobry rezultat – była to pierwsza kolumna w nim zadeklarowana.

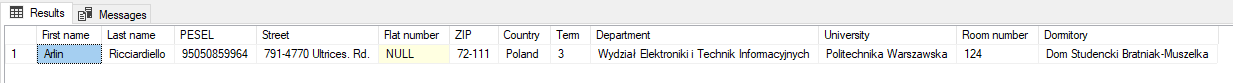
1. **Elementy programowalne**
   1. **Funkcje**

W celach zapoznawczych z elementami programowalnymi stworzone zostały funkcje: *StudentDataByPesel*, *NonFullRooms* oraz *StudentsSecurity.* Pierwsza z nich jako argument przyjmuje numer PESEL, dla którego zwraca dane osobowe studenta, pozyskane z utworzonej w punkcie 1. perspektywy.

Obraz zawierający tekst

Opis wygenerowany automatycznie

SELECT \* FROM StudentDataByPesel('95050859964')



Funkcja *NonFullRooms* przyjmuje jako argument ciąg znaków odpowiadający nazwie Domu Studenckiego. Jako rezultat zwracane są wszystkie niezapełnione pokoje w wybranym akademiku, wraz z informacją o liczbie wolnych miejsc.

Obraz zawierający tekst

Opis wygenerowany automatycznie

SELECT \* FROM NonFullRooms('Bratniak')

Obraz zawierający stół

Opis wygenerowany automatycznie

Zadaniem funkcji *StudentsSecurity* jest ograniczenie dostępu poszczególnych pracowników administracji Domów Studenckich do danych dotyczących placówki, w której pracują.

Obraz zawierający tekst

Opis wygenerowany automatycznie

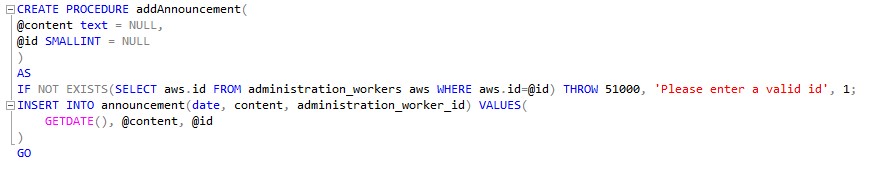
Poniżej przedstawiono rezultat pobrania studentów przez użytkownika będącego pracownikiem administracji Domu Studenckiego Bratniak-Muszelka:



* 1. **Procedury**

Istotną różnicą między funkcją, a procedurą jest fakt, że procedura nie musi zwracać wartości. W związku z tym może zostać wykorzystana do wprowadzania danych. Poniżej przedstawiono przykładowe implementacje procedur:

1. *addAnnouncement*  - procedura przyjmująca jako argumenty id pracownika administracji oraz treść ogłoszenia. Po sprawdzeniu, czy identyfikator jest prawidłowy, umieszcza nowe ogłoszenie w bazie danych

EXEC addAnnouncement @content=N'Przypominamy o obowiązku noszenia maseczek na terenie Domu Studenckiego', @id=1



1. *addIssue –* procedura przyjmująca jako argumenty treść skargi, id pracownika portierni oraz PESEL studenta, którego dotyczy skarga. Po sprawdzeniu, czy w bazie jest student o podanym numerze PESEL oraz pracownik portierni o podanym id, umieszcza nową skargę w bazie danych

Obraz zawierający tekst

Opis wygenerowany automatycznie

EXEC addIssue @content=N'Student zakłócał ciszę nocną', @doorkeeperId=1, @pesel='95081632112'

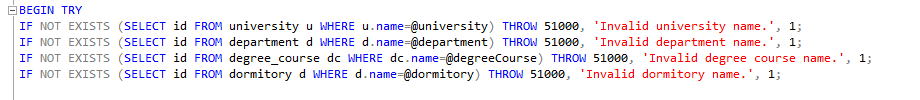


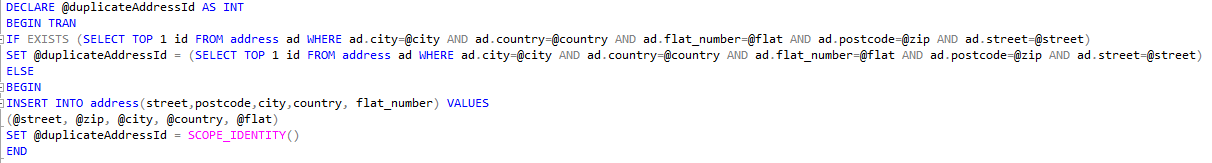
1. *addNewStudent* – procedura dodająca nowego studenta do systemu. Jako argumenty przyjmuje dane osobowe i adresowe studenta oraz kierunek studiów, Dom Studencki, w którym student będzie mieszkał oraz numer pokoju

Obraz zawierający tekst

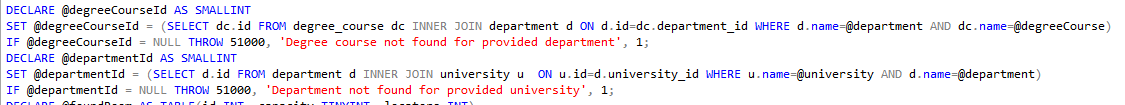
Opis wygenerowany automatycznie

Początkowym zadaniem procedury jest walidacja wprowadzonych pól. Następuje sprawdzenie, czy nazwy wydziału, kierunku studiów, uczelni oraz Domu Studenckiego są prawidłowe – czy występują w bazie danych.

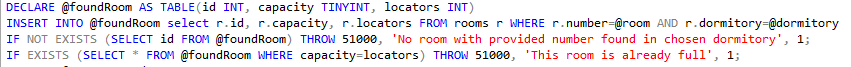
Po wstępnej walidacji następuje rozpoczęcie transakcji. Na podstawie wprowadzonych danych adresowych procedura sprawdza, czy podany adres nie istnieje już w bazie. Jeśli istnieje, zwraca jego identyfikator, w przeciwnym wypadku tworzy nowy adres.



Kolejnym krokiem jest sprawdzenie, czy kierunek studiów należy do podanego wydziału oraz czy wydział należy do podanej uczelni.



Należy również upewnić się, że pokój, do którego rejestrowany jest student należy do podanego Domu Studenckiego oraz że nie jest on już zapełniony.



Ostatnim krokiem jest zapisanie studenta do bazy danych oraz zakończenie transakcji.

Obraz zawierający tekst

Opis wygenerowany automatycznie

W przypadku błędu podczas realizacji procedury, transakcja zostanie przerwana i zwrócony zostanie odpowiedni komunikat.

Obraz zawierający tekst

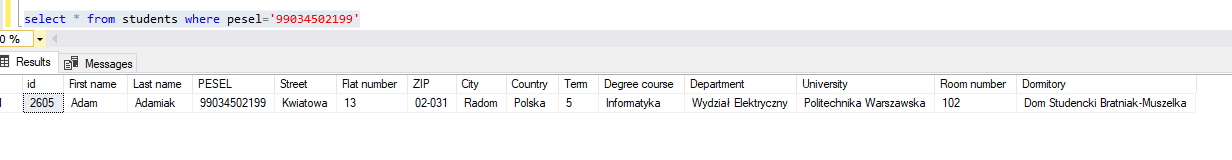
Opis wygenerowany automatycznieObraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznieObraz zawierający tekst

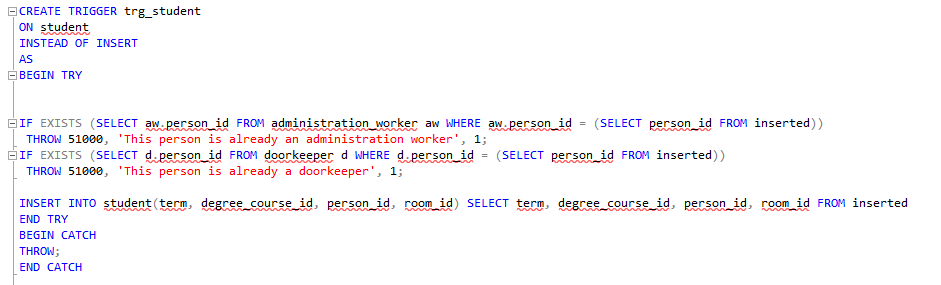
Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

* 1. **Wyzwalacze**

Generalizacja pozwala na uniknięcie duplikacji kolumn w różnych tabelach poprzez umieszczenie wspólnych danych w jednej tabeli. Rozwiązanie to ma pewien mankament logiczny – w opracowywanej przeze mnie bazie danych może nastąpić sytuacja, w której zarówno student, jak i pracownik administracji będzie tą samą osobą. Aby rozwiązać ten problem można utworzyćtriggery na wszystkich tabelach powiązanych z tabelą *person*. Każdy insert do tabel *student, administration\_worker, doorkeeper* zostanie zwalidowany – jeśli dana osoba będzie już miała przypisaną rolę, system nie wykona zapytania oraz zwróci błąd. Poniżej przedstawiono implementację wyzwalacza w przypadku tabeli *student*:



INSERT INTO student VALUES (7, 1, 2688, 1)

Obraz zawierający tekst

Opis wygenerowany automatycznie

Analogicznie, próba dodania pracownika portierni lub administracji w niewłaściwy sposób spowoduje wywołanie błędu:

INSERT INTO administration\_worker VALUES('test@mail', '123456788', 1, 300)

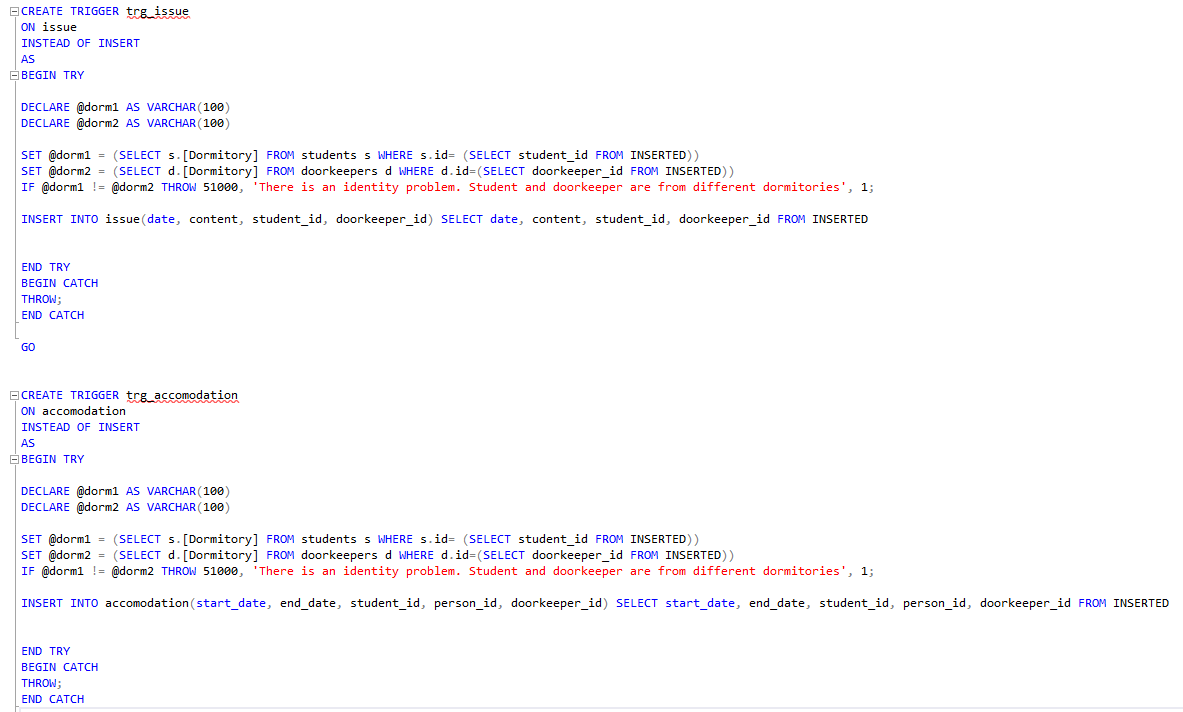
Obraz zawierający tekst

Opis wygenerowany automatycznie

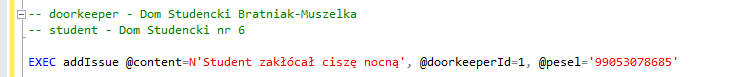
INSERT INTO doorkeeper VALUES(1, 2688)

Obraz zawierający tekst

Opis wygenerowany automatycznie

Podczas zgłaszania skargi bądź rejestracji noclegu należy podać identyfikatory studenta oraz pracownika portierni. Aby upewnić się, że dany student i pracownik portierni należą do tego samego Domu Studenckiego można wykorzystać wyzwalacze:

Obraz zawierający tekst

Opis wygenerowany automatycznie

1. **Automatyzacja zadań**

Za pomocą narzędzia Microsoft SQL Server Management Studio utworzony został Maintenance Plan, którego zadaniem jest generowanie backupu bazy danych codziennie o godzinie 21:00.

Obraz zawierający tekst

Opis wygenerowany automatycznieW celu przetestowania czy zadanie wykonuje się poprawnie, wywołano metodę *Execute***.** W rezultacie otrzymano plik .bak w folderze, w którym powinien znaleźć się backup bazy danych.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

W celu zapoznania z możliwościami narzędzia SQL Server Agent dodano do bazy danych nową tabelę – *payment* oraz odpowiadający jej widok - *payments*. W tabeli przechowywane będą płatności studentów za pobyt w Domach Studenckich. Poniżej przedstawiono przykładowe dane odczytane z widoku *payments:*

Obraz zawierający tekst, wewnątrz

Opis wygenerowany automatycznie

Biznesowym założeniem jest obowiązek uiszczenia opłaty przez studenta do 15. dnia każdego miesiąca. W celu skontrolowania terminów opłat w systemie utworzony został Job o nazwie *FindUnpaidBills*.

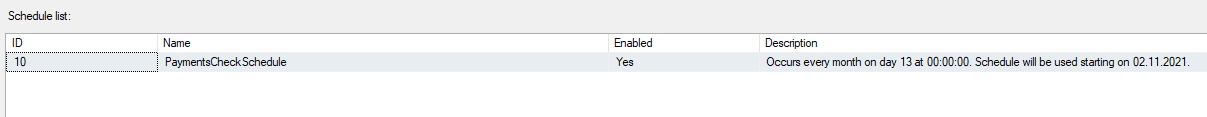
Obraz zawierający tekst

Opis wygenerowany automatycznie

Każdego 13. dnia miesiąca o godzinie 00:00 do bazy danych automatycznie kierowane jest następujące zapytanie:

select \* from students where id not in (select id from payments p where month(p.date)=month(getdate()) and year(getdate())=year(p.date))

Rezultat zapytania zapisywany jest do pliku tekstowego. Zawarte są w nim dane wszystkich studentów, którzy nie umieścili jeszcze opłaty za pobyt w Domu Studenckim w danym miesiącu.



Aby przetestować działanie zadania, chwilowo zmieniono datę wykonania zapytania na 02.11.2021 23:52.

Obraz zawierający tekst

Opis wygenerowany automatycznie



W rezultacie otrzymano dane studentów, którzy nie umieścili opłaty, w pliku tekstowym.



1. **Podsumowanie**
   1. Za pomocą widoków można udostępniać użytkownikom bazodanowym informacje z tabel powiązanych relacjami w postaci czytelnych i łatwo dostępnych zbiorów danych.
   2. Indeksy to struktury danych pozwalające na poprawę wydajności pozyskiwania danych z bazy. Należy jednak mieć na uwadze, że ich nieumiejętne wykorzystanie może spowodować więcej szkód niż dostarczyć korzyści.
   3. Za pomocą funkcji, procedur oraz wyzwalaczy można wdrożyć logikę biznesową do bazy danych oraz uprościć proces wyszukiwania i wprowadzania informacji do tabel.
   4. Microsoft SQL Server Management Studio umożliwia wdrożenie automatyzacji w bazie danych w prosty sposób, z wykorzystaniem narzędzia SQL Server Agent.