



BAZA DANYCH MPK

Dokument opisujący projekt zaliczeniowy kursu
„Bazy danych”, realizowany na WMII UJ 2021/22

Abstrakt

Prezentacja idei projektu, poczynionych założeń, użytych mechanizmów i własnych programów funkcyjnych, działających w DBMS SQL SERVER 2020
Opierając się na rozumowaniu tłumaczącym takie postępowanie i podającym konieczne do tego celu założenia (wykład), w dokumentacji słowo encja i tabela używa się wymiennie

Arkadiusz Biały, Jakub Steczkiewicz, Tomasz Szczepanik

1. Tematyka projektu

Baza danych reprezentuje urywek świata rzeczywistego (*Universe of Discourse*), będący przedsiębiorstwem komunikacyjnym, oferującym usługi pasażerom na określonym terenie. Projekt opierał się na obserwacjach dotyczących przede wszystkim krakowskiego Miejskiego Przedsiębiorstwa Komunikacji, lecz w naturalny sposób może odzwierciedlać dowolną firmę publiczną oferującą usługi transportowe, po dokonaniu odpowiedniej konwersji pojęć. Projekt ma w zamiarze umożliwienie przechowywania, modyfikowania oraz dodawania danych, ich selektywnego przeglądania i podejmowania na podstawie owych analiz dodatkowych akcji, ograniczanych przez mechanizmy kontrolujące poprawność zlecanych przez użytkownika transakcji, lecz ułatwionych, dzięki zredagowanym procedurom.

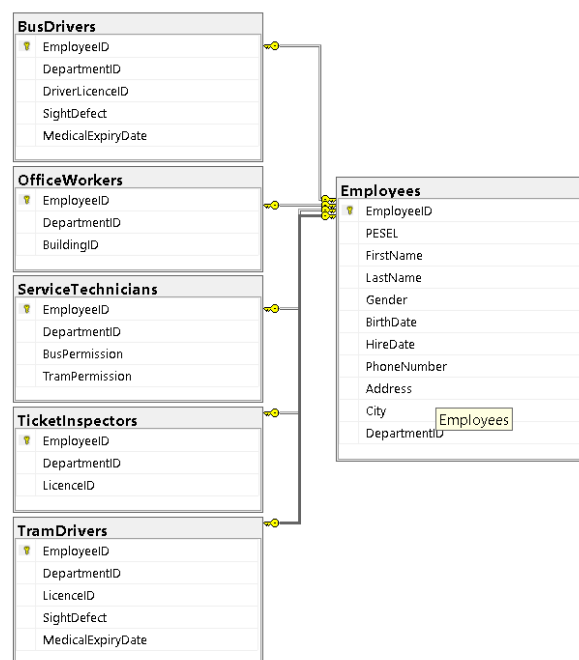
W bazie danych reprezentowane są dane o pracownikach przedsiębiorstwa, jego organizacji, infrastrukturze (również: zbiorze przystanków), taborze, dystrybucji i walidacji biletów oraz pewnych informacjach pobocznych, uzupełniających obraz przedmiotu badań.

2. Zbiór encji

W przedsiębiorstwie zatrudnieni są pracownicy, skierowani do pracy w poszczególnych działach. Należy wyróżnić kierowców i motorniczych, kontrolerów biletowych i mechaników, dbających o właściwy stan techniczny pojazdów, a także pracowników biurowych. Mnogość profesji znajduje odzwierciedlenie w zastosowanym schemacie dziedziczenia, w którym rodzicem wymienionych encji jest nadencja Pracownicy (*Employees*).

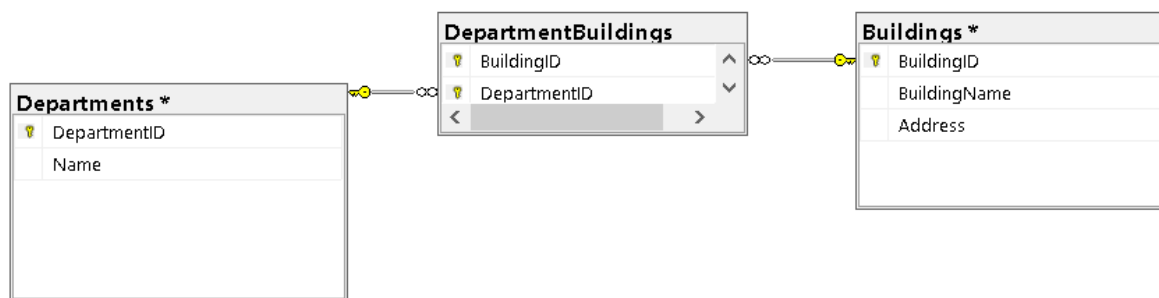
O każdym z pracowników zbierane są dodatkowe informacje, przechowywane w tabeli Urlopy (*EmployeeHolidays*) oraz Historia Pensji (*Salary History*), z przeznaczeniem do częstych aktualizacji w czasie użytkowania bazy danych.

Na infrastrukturę składają się przede wszystkim budynki (*Buildings*) należące do przedsiębiorstwa. W szczególny sposób projekt wyróżnia zajezdnie, przyznając im osobną tabelę (*Depots*) – ten zabieg znajduje uzasadnienie w chęci




zachowania praktycznej informacji dla obsługujących system, jaką jest (stałe) przypisanie każdego pojazdu do jednego z tych specjalnych obiektów architektonicznych.

Powracając do wcześniejszych uwag, dotyczących różnorodności profesji, jakie reprezentują pracownicy zakładu, w tabeli Działy (*Departments*) znajduje się odniesienie klucza obcego odpowiedniego atrybutu encji Pracownicy. Działy posiadają swoje organa w poszczególnych Budynkach – rozważywszy specyfikę zagadnienia, podjęta została decyzja o określeniu związku między powyższymi encjami jako relacja wiele do wielu (M:N). Naturalną konsekwencją poczynionego założenia musiała być tabela przejściowa Działy – Budynki (*DepartmentBuildings*), pozwalająca zarówno zajmować wiele budynków wskazanemu działowi, niewykluczająca równocześnie możliwości goszczenia różnych działów przez jeden i ten sam budynek.



Ewidencję sprzedaży biletów ułatwiają tabele dedykowane tej części działalności zakładu. Encja Typy Biletów (*TypesOfTickets*) zawiera spis wszystkich rodzajów biletów, umożliwiających skorzystanie z usług świadczonych przez przedsiębiorstwo. Znany z wielu implementacji podział biletów na okresowe (*PeriodicTickets*) i zwykłe uchwycono, korzystając powtórnie z mechanizmu dziedziczenia (*SoldSingleTickets*). W pierwszej grupie dodatkowo utworzono spis posiadaczy kart miejskich (informacja praktyczna) oraz listę kolejnych nabywanych przez klientów biletów (ważnych do danego dnia), natomiast druga grupa, zgodnie z nazwą, jest traktowana pod kątem bilansów sprzedaży odpowiednich biletów. Nasuwająca się w tym miejscu wątpliwość, związana z odległością czasową pomiędzy wprowadzeniem konkretnego biletu nieokresowego w obieg a jego nabyciem przez klienta

docelowego można usunąć w tych przypadkach poprzez liczenie sprzedaży od momentu zakupu puli biletów przez pośredników pierwszego stopnia.



AirReadings	
	AirReadingDate
	PM10
	PM25
	SO2
	NO2
	O3



Ceny biletów mogą być sankcjonowane przez różnorodne promocje i obniżki, dokumentowane w tabeli *Discounts*. Jedną z teoretycznych możliwości wprowadzenia jednodniowej, stuprocentowej redukcji ceny biletów może wynikać z odczytu stanu powietrza (potencjalnie pobieranego dobowo z wybranej stacji specjalistycznej), notowanego w Odczytach (Jakości) Powietrza, czyli tabeli

AirReadings. Stosowne symbole i oznaczenia chemiczne stanowią atrybuty mierzonych przez wspomniane stacje zanieczyszczeń.

Kolejna grupa encji opisuje pojazdy mechaniczne. Istotną różnicą zachodzącą pomiędzy modelami pojazdów (*VehicleModels*) oraz konkretnymi instancjami pojazdów (*Vehicles*) pozwala na wprowadzenie do systemu wielu pojazdów tego samego modelu. W tej ostatniej tabeli mieszczą się wszystkie pojazdy, co ma uzasadnienie projektanckie, tak jak i podział dziedziczny modeli na modele: autobusów, tramwajów i pojazdów specjalnych, figurujący w odpowiadających polskim nazwom wyszczególnionych tabelach. Pojazdy trafiające do napraw oddawane są w opiekę monterom.

Stanowiąca zbiór wierzchołków dla grafu, jakim jest sieć połączeń tramwajowo – autobusowych, tabela Przystanki (*Stops*) została uzupełniona wszystkimi przystankami aglomeracji znajdującymi się na stronie internetowej krakowskiego MPK (dostęp: I kwartał 2022). Zamieszczone w bazie linie zapisywane są jako zbiór rekordów o stałym (w obrębie linii) polu „linia” (*LineId*) oraz dwóch polach enumerujących kolejne przystanki tworzące trasę przemierzaną przez wehikuły podróżujące pod szyldem zdobionym tym numerem. Tabela Kursy wskazuje na linię obsługującą kurs, typ kursu (więcej w sekcji poświęconej ograniczeniom) i godzinę odjazdu z przystanku początkowego. Problem wyznaczania przyjazdów na poszczególne przystanki został rozwiązany dzięki tabeli krawędzi grafu – połączeń międzyprzystankowych z zapisaną długością (czasową), dzielącą każdą parę (nieuporządkowaną) połączonych wierzchołków; oczywiście osobno dla autobusów i osobno dla tramwajów. W

BusConnections	
	FromStopID
	ToStopID
	Duration

TramConnections	
	FromStopID
	ToStopID
	Duration

konkretnych kursach (*DetailedBus*- i *TramCourses*) przygotowane są wszystkie przejazdy w nadchodzącym przedziale czasu (np. na obecny tydzień), którym należy przydzielić kierowców i motorniczych wraz z odpowiednim taborem.

3. Ograniczenia nałożone na dane

Ograniczenia danych należy podzielić na wynikające z natury rzeczy i systemu definicji oraz na znajdujące swoje uzasadnienie w logice biznesowej. Do pierwszych należą te rodzaju:

- nieujemność przebiegu pojazdu (*mileage*)
- następstwo daty rozpoczęcia i zakończenia urlopu ($DateTo \geq DateFrom$)
- następstwo daty rozpoczęcia i zakończenia produkcji modelu (analogicznie)
- długość i forma numeru PESEL

Na wycinek rzeczywistości zostały w trakcie projektu narzucone również inne ograniczenia oraz konwencje służące do zapisu i korzystania z informacji. Do tych należą z kolei:

- numery licencji zezwalające na wykonywanie zawodu przez reprezentantów poszczególnych fachów (np. *LicenceID*, *DriverLicenceID*)
- przypisanie tylko jednego mechanika do wykonania naprawy pojazdu (np. tylko kierownik podpisuje raport)
- niedublowanie się nazw przystanków, np. w sąsiednich gminach (choć w dalszym ciągu w każdym razie będą rozróżniane personalizowanymi identyfikatorami)
- pustość teoriomnogościowego przecięcia dowolnych dwóch zbiorów kursów, należących do dwóch różnych, ustalonych dni – w projekcie czyni się milczące założenie, że żaden kurs nie może kończyć się innego dnia, niż się rozpoczął (doświadczenie pokazuje, że schemat stosowany przez MPK w Krakowie nie narusza w sposób zasadniczy tego założenia)
- zdolność każdego kierowcy i motorniczego do prowadzenia każdego modelu pojazdu w przynależnej im kategorii (odpowiednio: autobusów i tramwajów; może nie być zgodne ze stanem faktycznym, choć taka ewentualność nie ma charakteru nieuchronnej)
- student ubiegający się o taki status w rejestrze posiadaczy karty miejskiej nie może mieć więcej niż 26 lat, tak jak emeryt – mniej niż 60, zaś uczeń – więcej niż 18

- jeżeli dany model pojazdu nie został wycofany z produkcji, w bazie danych data jej końca zakończenia widnieje pod flagą NULL
- różne ograniczenia pomijalne, np. długość maksymalna noty relacjonującej przebieg naprawy pojazdu,
- ograniczenia czysto implementacyjne, np. typ kursów zawierający się w jednowyrazowych podciągach akronimu RSH (*regular* – kurs w dzień powszedni, *saturday* – kurs sobotni, *holiday* – kurs w dzień świąteczny)

4. Dodatkowe więzy integralności

W tym punkcie należy wymienić zależność pomiędzy encjami linii tramwajowych i autobusowych, a encjami połączeń (także tramwajowych oraz autobusowych). Ponieważ, z uwagi na redundancję danych i estetykę przechowywania zrezygnowano z przechowywania w pierwszych wymienionych tabelach par przystanków w jednym wierszu w jakiegokolwiek formie (np. obecny przystanek – następny przystanek), a właśnie to rozwiązanie posłużyło do zunifikowania czasów przejazdów pomiędzy dwoma określonymi przystankami (czas ten jest taki sam dla każdej linii autobusowej i osobno, dla każdej linii tramwajowej), wobec tego w schemacie baz danych istnieje relacja zachodząca pomiędzy identyfikatorami każdego dwóch kolejnych przystanków tej samej linii a ową parą w tabeli połączeń.

BusLines	
LineID	
StopNumber	
StopID	
BusConnections	
FromStopID	
ToStopID	
Duration	

5. Indeksy

W bazie danych nie powstały indeksy, nie znaleziono dla nich koniecznego zastosowania.

6. Koncepcja pielęgnacji – kopia zapasowa

Konserwacji bazy danych można dokonywać, korzystając z polecenia BACKUP DATABASE. Używając odpowiednich opcji możliwe jest zredukowanie czasu potrzebnego na kolejne modyfikacje kopii zapasowej czy zwiększenie bezpieczeństwa powstającego pliku (flaga ENCRYPTION). Kopię zapasową należy aktualizować regularnie – przykładowo: raz na tydzień.

```
BACKUP DATABASE MPK
TO DISK = 'D:\MPK\testDB.bak'
WITH DIFFERENTIAL;
```

7. Widoki

Widoki pozwalają ograniczyć dostęp do danych wrażliwych użytkownikom bazy danych posiadającym mniejsze uprawnienia. Okazują się też być pomocne przy zapytaniach pośrednich, które często mogą występować w bardziej zaawansowanych sformułowaniach.

1. Widok kierowców autobusów – Bus_Drivers przekazuje informacje o kierowcach autobusów, zawarte w tabeli Pracownicy, zatem nadrzędnej w stosunku do kierowców autobusów i przechowującej informacje właściwe każdej profesji pracowniczej

```
CREATE VIEW Bus_Drivers
AS
    SELECT E.FirstName AS [First Name], E.LastName AS [Last Name], D.DriverLicenceID AS [Licence ID]
    FROM Employees AS E JOIN BusDrivers AS D
    ON E.EmployeeID = D.EmployeeID
    WHERE E.DepartmentID = 1
    ORDER BY [Last Name], [First Name]
    OFFSET 0 ROWS
```

2. Widok kierowców tramwajów – Tram_Drivers widok analogiczny i również potrzebny jak poprzedni

```
CREATE VIEW Tram_Drivers
AS
    SELECT E.FirstName AS [First Name], E.LastName AS [Last Name], D.LicenceID AS [Licence ID]
    FROM Employees AS E JOIN TramDrivers AS D
    ON E.EmployeeID = D.EmployeeID
    WHERE E.DepartmentID = 2
    ORDER BY [Last Name], [First Name]
    OFFSET 0 ROWS
```

3. Widok Ticket_Types - widok rodzajów biletów – generuje najważniejsze informacje (podstawowe) z tabeli Typy biletów

```
CREATE VIEW Ticket_Types
AS
    SELECT Name AS [Ticket Name], Price FROM TypesOfTickets
```

4. Widok Vehicle_models – widok modeli i ich stopnia implementacji w tabor przedsiębiorstwa komunikacyjnego – prócz standardowych informacji na temat modelu podaje liczbę pojazdów sygnowanych tym numerem, należących do zakładu (kod źródłowy na następnej stronie)

```
CREATE VIEW Vehicle_Models
AS
    SELECT M.ModelName, COUNT(*) AS Quantity,
    CASE
        WHEN M.ModelID IN (SELECT ModelID FROM TramModels) THEN 'Tram'
        WHEN M.ModelID IN (SELECT ModelID FROM BusModels) THEN 'Bus'
        ELSE 'Other'
    END AS Vechicle_Type
    FROM Vehicles AS V JOIN VehicleModels AS M
    ON V.ModelID = M.ModelID
    GROUP BY M.ModelName, M.ModelID
    ORDER BY M.ModelID
    OFFSET 0 ROWS
```

5. Widok Passengers_With_Periodic_Tickets, zwracający kompleksowy zestaw danych dotyczący pasażerów będących właścicielami okresowych

```
CREATE VIEW Passengers_With_Periodic_Tickets
AS
    SELECT * FROM Passengers P JOIN PeriodicTickets PT ON P.PassengerID = PT.OwnerID
```

biletów komunikacyjnych

6. Widok `Single_Tickets_Sold_Each_Day`, będący raportem sprzedaży biletów czasowych, pogrupowanych względem dnia sprzedaży (użycie

```
CREATE VIEW Single_Tickets_Sold_Each_Day
AS
    SELECT DateOfPurchase, COUNT(*) AS TicketsSold FROM SoldSingleTickets
    GROUP BY DateOfPurchase
```

funkcji agregującej `count()`)

7. Widok `Days_With_Discounts` stanowiący wyjaśnienie części stosowanych zniżek – wskazuje dni, w których, z uwagi na przekroczone normy zanieczyszczenia powietrza, w zgodzie z odpowiednią ustawą, komunikacja publiczna będzie darmowa

```
CREATE VIEW Days_With_Discounts
AS
    SELECT AirReadingDate FROM AirReadings WHERE PM10 >= 150 OR PM25 >= 150 OR NO2 >= 150 OR SO2 >= 150 OR O3 >= 150
GO
```

8. Widok `Vehicles_in_depots` prezentuje spis wszystkich zajezdni (*Depots*) wraz z liczbą konkretnych modeli w nich stacjonujących (klauzula `GROUP BY`) – choć z punktu widzenia rozważań teoretycznych może nie być to decydujący przykład, to widok ten demonstruje podstawową funkcję konstruktów `VIEWS` – tworzy istotnie inną perspektywę na ten sam zestaw danych – grupa korzystających z systemu może nie być zainteresowana dywagacją, które konkretnie pojazdy przypisane zostały do której zajezdni – z ich punktu widzenia, o wiele ważniejszy może być raport, ile i jakich modeli znajduje się we wskazanych miejscach.

```
CREATE VIEW Vehicles_in_depots
AS
    SELECT B.Address, B.BuildingName AS Depot, VM.ModelName, COUNT(VM.ModelName) AS Count
    FROM Depots AS D JOIN Vehicles AS V
    ON D.DepotID = V.DepotID
    JOIN Buildings AS B
    ON D.BuildingID = B.BuildingID
    JOIN VehicleModels AS VM
    ON VM.ModelID = V.ModelID
    GROUP BY B.BuildingName, B.Address, VM.ModelName
    ORDER BY Address, Depot, Count DESC
    OFFSET 0 ROWS
```

8. Wyzwalacze

1. Wyzwalacz New_Discounts – zdefiniowany dla tabeli AirReadings: po dodaniu odczytu powietrza wyzwalacz sprawdza, czy można wygenerować automatyczną obniżkę cen biletów, działającą ze skutkiem natychmiastowym.
2. Wyzwalacz Passengers_Data_Check_Insert weryfikuje sensowność danych podawanych do tabeli Passengers – sprawdza ich wewnętrzną spójność. Tryb INSTEAD OF INSERT pozwala wprowadzić jedynie te rekordy, które nie są wewnętrznie sprzeczne. Kluczowe jest zastosowanie kursora do przeglądania tabeli INSERTED, natomiast, poglądowo, fragment:

```
SELECT @years = DATEDIFF(yy, @birthDate, GETDATE()) - CASE WHEN (MONTH(@birthDate) > MONTH(GETDATE()))
SET @gender = (SELECT LEFT(RIGHT(CONVERT(VARCHAR, @pesel), 2), 1))
IF (@gender = '0' OR @gender = '2' OR @gender = '4' OR @gender = '6' OR @gender = '8')
BEGIN
    IF (@years < 60 AND @pensioner = 1)
    BEGIN
        SET @errorMsg = @firstName + ' ' + @lastName + ' cannot be a pensioner.'
        ;THROW 51001, @errorMsg, 1
    END
END
```

wylicza i pobiera stosowne parametry i następnie deklaruje, że emerytem nie może być osoba poniżej sześćdziesiątego roku życia nawet wówczas, gdy dane przepisane do bazy danych by na to wskazywały

3. Wyzwalacz Passengers_Data_Check_Update – aplikuje symetryczne działanie co poprzednik, lecz jest wywoływany na skutek nie wstawienia nowych elementów, lecz modyfikowania tych starych. Fragment kodu wyzwalacza usuwa możliwość przyjęcia za poprawne danych osoby, która podała się za studenta (wartość pola oznaczona bitową flagą 1), lecz ma więcej niż 26 lat – system rzuci wówczas odpowiedni błąd.

```

IF (@years > 26 AND @student = 1)
BEGIN
    SET @errorMsg = @firstName + ' ' + @lastName + ' cannot be a student.'
    ;THROW 51002, @errorMsg, 1
END

```

4. Wyzwalacz deskryptowy Insert_TramConnections – wstawienie połączenia tramwajowego wiąże się z podaniem długości połączenia, która nie może być ujemna. Użytkownik nieświadomy takiej zależności może, przy nieszczęśliwie wprowadzonych danych, przekonać się, że pole to nie przyjmie wartości ujemnych – pracownik otrzyma stosowne wyjaśnienie, a dane nie zostaną wprowadzone.

```

print CAST(@From AS VARCHAR) + ', ' + CAST(@To AS VARCHAR) + ', ' + CAST(@Time AS VARCHAR) + ' will not be added due to non-positive journey duration between the stops'

```

5. Wyzwalacz Insert_BusConnections – wyzwalacz analogiczny do poprzednika (fragment poniżej)

```

CREATE TRIGGER Insert_BusConnections
ON BusConnections
INSTEAD OF INSERT
AS
BEGIN

    DECLARE @From INT
    DECLARE @To INT
    DECLARE @Time INT

    DECLARE Records CURSOR
        FOR SELECT FromStopID, ToStopID, Duration FROM Inserted
        FOR READ ONLY

```

9. Funkcje

1. Funkcja Interval_Air_Readings generuje skondensowany raport w trzech skalach czasowych wraz z wykorzystaniem aparatu matematycznego do odczytów (odchylenie standardowe i średnia arytmetyczna. Zwraca trzy krotki tworzące tabelę

```

CREATE FUNCTION Interval_Air_Readings (@date DATE)
RETURNS @table TABLE (
    Interval NVARCHAR(50),
    Average_PM10 DECIMAL(7,3),
    Standard_Deviation_PM10 DECIMAL(7,3),
    Average_PM25 DECIMAL(7,3),
    Standard_Deviation_PM25 DECIMAL(7,3),
    Average_SO2 DECIMAL(7,3),
    Standard_Deviation_SO2 DECIMAL(7,3),
    Average_NO2 DECIMAL(7,3),
    Standard_Deviation_NO2 DECIMAL(7,3),
    Average_O3 DECIMAL(7,3),
    Standard_Deviation_O3 DECIMAL(7,3)
)
AS
BEGIN

```

Kolumny bezpośrednio implikują zawartość tej tabeli (przedziały czasowe to ostatni: tydzień, miesiąc, rok).

2. Funkcja Get_Route (@line INT) zwraca trasę podanej argumentem linii, składającej się z kolejnych nazw przystanków w jednej z dwóch wybranych kolejności. Poniżej zasadnicza część funkcji – dzięki operatorowi łączenia tabel JOIN prócz identyfikatorów przystanków, operator pozna również ich nazwy.

```

BEGIN
    INSERT INTO @route
        SELECT TL.StopNumber, S.StopName
        FROM Stops AS S JOIN TramLines AS TL
        ON TL.StopID = S.StopID
        WHERE TL.LineID = @line
        ORDER BY TL.StopNumber

END

ELSE
    BEGIN
        INSERT INTO @route
            SELECT BL.StopNumber, S.StopName
            FROM Stops AS S JOIN BusLines AS BL
            ON BL.StopID = S.StopID
            WHERE BL.LineID = @line
            ORDER BY BL.StopNumber

        END
    END

RETURN

```

Przykładowe zastosowanie funkcji korzysta z porządkowania wyjścia zgodnie z rosnącym numerem przyporządkowanym konkretnym przystankom.

```

SELECT * FROM dbo.Get_Route(52)
ORDER BY Ord

```

3. Funkcja Number_Of_Types_Of_Drives() zwraca jedną, lecz niezwykle ważką krotkę – w kolejnych kolumnach znajdzie się liczba modeli

autobusów o określonym napędzie – spalinowym, elektrycznym czy też hybrydowym.

```
CREATE FUNCTION Number_Of_Types_Of_Drives()
RETURNS @table TABLE
(
    Combustion INT,
    Electric INT,
    Hybrid INT
)
AS
BEGIN

    DECLARE @noOfCombustion INT
    DECLARE @noOfElectric INT
    DECLARE @noOfHybrid INT

    SET @noOfCombustion = (SELECT COUNT(*) FROM (SELECT * FROM BusModels WHERE Drive = 'Combustion') AS subquery)
    SET @noOfElectric = (SELECT COUNT(*) FROM (SELECT * FROM BusModels WHERE Drive = 'Electric') AS subquery1)
    SET @noOfHybrid = (SELECT COUNT(*) FROM (SELECT * FROM BusModels WHERE Drive = 'Hybrid') AS subquery2)

    INSERT INTO @table
        SELECT @noOfCombustion, @noOfElectric, @noOfHybrid

    RETURN

END

GO
```

Rozwiązanie korzysta z funkcji COUNT() oraz z prostego zapisania do zmiennej zapytań opartych na odpowiednich warunkach WHERE.

4. Funkcja Number_Of_Vehicles() pełni analogiczną rolę jak poprzednia, lecz – tym razem – zwraca liczbę pojazdów. Poniżej zamieszczono pełny kod, wraz z przykładem zastosowania.

```
CREATE FUNCTION Number_Of_Vehicles()
RETURNS @table TABLE
(
    Buses INT,
    Trams INT,
    SpecialVehicles INT
)
AS
BEGIN

    DECLARE @noOfTrams INT
    DECLARE @noOfBuses INT
    DECLARE @noOfSpecialVehicles INT

    SET @noOfTrams = (SELECT COUNT(*) FROM (SELECT V.ModelID FROM VehicleModels V JOIN TramModels T ON V.ModelID = T.ModelID) AS subquery)
    SET @noOfBuses = (SELECT COUNT(*) FROM (SELECT V.ModelID FROM VehicleModels V JOIN BusModels B ON V.ModelID = B.ModelID) AS subquery1)
    SET @noOfSpecialVehicles = (SELECT COUNT(*) FROM (SELECT V.ModelID FROM VehicleModels V JOIN SpecialVehicleModels S ON V.ModelID = S.ModelID) AS subquery2)

    INSERT INTO @table
        SELECT @noOfBuses, @noOfTrams, @noOfSpecialVehicles

    RETURN

END

GO

SELECT * FROM Number_Of_Vehicles()
```

10. Procdedury składowane

1. Procedura Passenger_Ticket_Info daje spersonalizowany wgląd w dane konkretnego klienta oraz – w przypadku, gdy klientowi przysługuje dodatkowa zniżka – informuje go o tym fakcie. Przegląd danych odbywa się przy pomocy kursora, natomiast przed zwróceniem danych, tekst przechodzi proces właściwego formatowania.

```
CREATE PROCEDURE Passenger_Ticket_Info (@id INT) AS
BEGIN
    IF NOT EXISTS(SELECT * FROM Passengers P WHERE P.PassengerID = @id)
    BEGIN
        ;THROW 51000, 'Passenger does not exist.', 1
    END

    DECLARE @pesel CHAR(11)
    DECLARE @passengerID INT
    DECLARE @birthDate DATE
    DECLARE @firstName VARCHAR(50)
    DECLARE @lastName VARCHAR(50)
    DECLARE @placeOfResidence VARCHAR(50)
    DECLARE @city VARCHAR(50)
    DECLARE @student BIT
    DECLARE @pupil BIT
    DECLARE @honoraryBloodDonor BIT
    DECLARE @pensioner BIT

    DECLARE cur CURSOR FOR (SELECT * FROM Passengers P WHERE P.PassengerID = @id)
    OPEN cur
    FETCH NEXT FROM cur INTO @passengerID, @pesel, @birthDate, @firstName, @lastName, @placeOfResidence, @city, @student, @pupil,
    @honoraryBloodDonor, @pensioner
    SELECT FORMATMESSAGE('Passenger Info: PESEL: %s; ID: %d; Date of birth: %s; first name: %s; last name: %s; address: %s; city: %s; student: %d; pupil: %d; |
    @lastName, @placeOfResidence, @city, CONVERT(INT, @student), CONVERT(INT, @pupil), CONVERT(INT, @honoraryBloodDonor), CONVERT(INT, @pensioner))
    CLOSE cur
    DEALLOCATE cur

    IF (@student = 1 OR @pupil = 1 OR @honoraryBloodDonor = 1 OR @pensioner = 1)
        SELECT FORMATMESSAGE('%s %s with passenger id %d can buy tickets with reduced prices.', @firstName, @lastName, @id)
    ELSE
        SELECT FORMATMESSAGE('%s %s with passenger id %d cannot buy tickets with reduced prices.', @firstName, @lastName, @id)
END
```

2. Procedura Passengers_With_Valid_Periodic_Tickets wypisuje listę pasażerów MPK, którzy dla zadanej w argumencie daty mają ważne bilety okresowe. Wewnątrz procedury tworzona jest tabela, uzupełniana wraz z przemieszczaniem się kursora i dokonywanych obliczeń, uzależnionych od typów posiadanych biletów.

```

CREATE PROCEDURE Passengers_With_Valid_Periodic_Tickets (@currentDay DATE) AS
BEGIN
    DECLARE cur CURSOR FOR SELECT PT.TicketID, OwnerID, DateFrom, PassengerID, FirstName, LastName FROM PeriodicTickets PT JOIN TypesOfTickets T ON
    JOIN Passengers P ON PT.OwnerID = P.PassengerID

    DECLARE @result Table(ID INT, FirstName VARCHAR(50), LastName VARCHAR(50), DateOfPurchase DATE)

    OPEN cur
    DECLARE @ticketID INT
    DECLARE @ownerID INT
    DECLARE @dateFrom DATE
    DECLARE @days INT
    DECLARE @isValid BIT
    DECLARE @firstName VARCHAR(50)
    DECLARE @lastName VARCHAR(50)
    DECLARE @id INT
    FETCH cur INTO @ticketID, @ownerID, @dateFrom, @id, @firstName, @lastName
    DECLARE @daysDiff INT

    WHILE (@@FETCH_STATUS = 0)
    BEGIN
        IF (@dateFrom <= @currentDay)
        BEGIN
            SET @daysDiff = DATEDIFF(day, @dateFrom, @currentDay)
            IF ((@ticketID BETWEEN 18 AND 21 AND @daysDiff <= 30) OR (@ticketID BETWEEN 22 AND 23 AND @daysDiff <= 180))
            INSERT INTO @result VALUES(@ownerID, @firstName, @lastName, @dateFrom)
        END
        FETCH cur INTO @ticketID, @ownerID, @dateFrom, @id, @firstName, @lastName
    END

    CLOSE cur
    DEALLOCATE cur

    SELECT * FROM @result
END

```

3. Procedura Delete_Passengers_With_Invalid_Periodic_Tickets działa analogicznie do poprzedniczki, tyle że usuwa bilety nieważne danego (argumentem) dnia.
4. Procedura Ticket_Prices zwraca ceny biletów, które obowiązują danego dnia. Czyni to, analizując listę potencjalnych obniżek.

```

CREATE PROCEDURE Ticket_Prices(@day DATE) AS
BEGIN
    SELECT * INTO #result FROM TypesOfTickets

    IF EXISTS (SELECT * FROM Days_With_Discounts
    WHERE @day = Days_With_Discounts.AirReadingDate)
        UPDATE #result SET Price = 0 WHERE Periodic = 0

    SELECT * FROM #result
END
GO

```

5. Procedura Add_Employee dodaje pracownika, podnosząc odpowiednie błędy, w zależności od rodzaju błędu, komunikując się z użytkownikiem bazy danych. Procedura osobno sprawdza znaczniki NULL, osobno zaś – pojawienie się wyjątków w drodze wstawienia danych.
6. Procedura Add_Bus_Driver korzysta z procedury dodawania pracownika, używając jako flagi błędu – argumentu wyjściowego @error. W ten

sposób dowiaduje się, czy powstał na drodze rekordu błąd, czy też można bezpiecznie kontynuować wstawianie.

7. Procedura Add_TramDriver – analogiczna do procedury Add_Bus_Driver.
8. Procedura Add_Technician – analogiczna do procedury Add_Bus_Driver.
9. Procedura Add_Inspector – analogiczna do procedury Add_Bus_Driver.
10. Procedura Add_Office_Worker – analogiczna do procedury Add_Bus_Driver.
11. Procedura Add_Vehicle stara się dodać egzemplarz samochodu, lecz nie bezwzględnie – wpierw sprawdza więzy z tabelą modeli, określoność rekordów, przebieg pojazdu i rok produkcji. W razie niepomyślności – procedura informuje użytkownika.
12. Procedura AddStopToBusLineBeforeAnother dodaje do linii @line przystanek o identyfikatorze @id i umieści go przed przystankiem o identyfikatorze @id (równym zero, gdy przystanek ma zostać wstawiony na koniec trasy pewnej, obecnie kursującej linii. Pomysł realizacji opiera się na zrobieniu miejsca na rekord, poprzez zwiększenie numeru porządkowego przystanków, które, po procedurze, wystąpią po dodawanym przystanku.
13. Procedura AddStopToTramLineBeforeAnother działa analogicznie do poprzedniej.