BAZA DANYCH MPK

Adam Keklak

Baza Danych MPK

Baza Danych

Projekt przedstawia przykładową bazę danych miejskiego przedsiębiorstwa komunikacyjnego.

Projekt umożliwia przechowywanie, dodawanie, modyfikacje oraz przeszukiwanie znajdujących się w niej danych. Ułatwia ona również zarządzanie danymi. W skład bazy wchodzą informacje o pracownikach, ich urlopy, bilety, historia ich zakupów, oraz informacje o kursach i pojazdach którymi dysponuje przedsiębiorstwo.

W bazie znajdują się różne dane wszystkich pracowników, ich działów oraz ich zawód. W innych tabelach znajdują się dodatkowe informacje o nich takie jak historia wypłat, historia urlopów.

W projekcie znajdują się informacje o pojazdach, które są do dyspozycji z informacją o ostatnim przeglądzie, dwie tablice dziedziczące po niej.

Kolejna tabela przedstawia rodzaje biletów, czas ich trwania oraz ceny. Przechowywane są też informacje o historii sprzedaży biletów.

Ostatnią grupą są linie komunikacyjne, przystanki oraz tabela przechowywująca trojki numer linii, numer przystanku oraz informacja o tym, którym w kolejności jest ten przystanek na danej linii.

Ograniczenia

Zakładam, że wszystkie dane są wprowadzone poprawnie.

Wszystkie linie kursują we wszystkie dni tygodnia.

Każda linia kursuje tylko w jedną stronę.

Widoki

1. Wypisywanie kierowców

```
CREATE VIEW Drivers
AS
    SELECT FirstName ,LastName
    FROM Employees
    WHERE DepartmentID = 'Driver'
```

2. Informacje o różnych rodzajach biletów

```
CREATE VIEW TicketTypes
AS
    SELECT Name, Price FROM Tickets
```

3. Ilość pojazdów poszczególnych rodzajów

```
CREATE VIEW Vehicle_Types
AS
    SELECT VehicleType, COUNT(*) AS Quantity
    FROM Vehicles LEFT JOIN Buses ON Vehicles.VehicleID = Buses.VehicleID
    LEFT JOIN Trams ON Vehicles.VehicleID = Trams.VehicleID
    GROUP BY VahicleType
```

4. Nowi pracownicy (<2 miesiące)

```
CREATE VIEW NewHires
AS
    SELECT * FROM Employees
    WHERE HireDate > GETDATE() - DATEADD(MONTH,-2,GETDATE())
```

5. Profit ze sprzedaży biletów w ostatnim miesiącu

```
CREATE VIEW TicketsLastMonth
AS
    SELECT SUM(Price) AS Profit FROM TicketHistory
    WHERE HireDate > GETDATE() - DATEADD(MONTH,-1,GETDATE())
```

6. Ilość zużytych dni wolnych poszczególnych pracowników

```
CREATE VIEW NumberOfUsedHolidasy
AS
    SELECT EmployeeID ,SUM(DATEDIFF(EndDate, StartDate)) AS UsedHolidays FROM Holidays
```

7. Historyczny profit przedsiębiorstwa

```
CREATE VIEW Profit
AS
    SELECT (SELECT SUM(price) FROM TicketHistory - SELECT SUM(salary) FROM SalaryHistory) AS Profit
```

8. Dochód poszczególnych pracowników podczas pracy

```
CREATE VIEW SalaryOfEmployee
AS
    SELECT EmployeeID, SUM(salary) AS Salary FROM SalaryHistory
```

9. Ilość linii, które obsługują przystanki

```
CREATE VIEW PopularityOfStop
AS
    SELECT Stops.Name, SUM(*) AS NumberOfLinesConnected FROM Stops JOIN Connections ON Stops.StopID = Connections.StopID
```

10. Dniowy dochód ze sprzedaży biletów

```
CREATE VIEW TicketSalesByDay
AS
    SELECT TicketsHistory.Time AS Time , SUM(price) AS Profit FROM TicketsHistory
    GROUP BY TicketsHistory.Time
```

Wyzwalacze

1. Ustawia ostatnią inspekcje na datę dodania go do tabeli

```
CREATE TRIGGER VehicleInsert
ON Vehicles
AFTER INSERT
AS
BEGIN
    UPDATE inserted
    SET Lastinspected = GETDATE()
    WHERE VehicleID IN (SELECT VehicleID FROM inserted)
END
```

2. Po dodaniu nowej linii podnosi cenę biletów

```
CREATE TRIGGER OnNewLine
ON Lines
AFTER INSERT
AS
BEGIN
    UPDATE Tickets
    SET Price = Price * 1.05
END
```

3. Gdy pensje w danym miesiącu jest za duża obniża wypłatę zarządu

```
CREATE TRIGGER DecreaceSalary
ON SalaryHistory
AFTER INSERT
AS
BEGIN
    IF(SELECT SUM(Salary) FROM inserted > 500000)
    BEGIN
        UPDATE Employees
        SET Salary = Salary * 0.80
        WHERE Position IN ('Boss', 'Managment')
    END
END
```

4. Blokuje rozpoczęcie urlopu podczas gdy inny pracownik już ma w tym terminie urlop

```
CREATE TRIGGER NoHoliday
ON Holidays
INSTEAD OF INSERT
AS
BEGIN
    DECLARE curs CURSOR FOR SELECT * FROM inserted
    OPEN curs
    DECLARE @ID INT
    DECLARE @Sdate DATE
    DECLARE @Edate DATE
    FETCH curs INTO @ID, @Sdate, @Edate

    WHILE (@@FETCH_STATUS = 0)
    BEGIN

        IF(NOT EXISTS(SELECT * FROM Holidays WHERE @Sdate BETWEEN StartDate AND EndDate))
        BEGIN
        INSERT INTO Holidays VALUES(@ID, @Sdate, @Edate)
        END
        FETCH cur INTO @ID, @Sdate, @Edate
    END


    CLOSE curs
    DEALLOCATE curs

END
```

5. Wypisuje zwolnionych pracowników

```
CREATE TRIGGER FireEmployee
ON Employees
AFTER DELETE
AS
BEGIN

    DECLARE curs CURSOR FOR SELECT FirstName, LastName FROM deleted
    OPEN curs
    DECLARE @Fname NVARCHAR(50)
    DECLARE @Lname NVARCHAR(50)
    FETCH curs INTO @Fname, @Lname

    WHILE (@@FETCH_STATUS = 0)
    BEGIN

        PRINT 'Employee' + @Fname + ' ' + @Lname + ' got fired'
        FETCH curs INTO @Fname, @Lname
    END
END
```

Procedury

1. Sprawdza poprawność nowych pracowników

```
CREATE PROCEDURE Add_Employee(
 @ID INT = NULL,
 @first_name VARCHAR(50) = NULL,
 @second_name VARCHAR(50) = NULL,
 @PESEL CHAR(11) = NULL,
 @gender NVARCHAR(50) = NULL,
 @birth_date DATE = NULL,
 @hire_date DATE = NULL,
 @salary MONEY = NULL,
 @phone_number CHAR(9),
 @address VARCHAR(50) = NULL,
 @department VARCHAR(50) = NULL,
 @position VARCHAR(50) = NULL,
 @out BIT OUTPUT
)
AS
DECLARE @error_msg NVARCHAR(100)
    IF @ID is NULL OR @first_name IS NULL OR @second_name IS NULL OR @PESEL IS NULL OR @gender IS NULL
        OR @birth_date IS NULL OR @hire_date IS NULL OR @salary IS NULL OR @address IS NULL
        OR @department IS NULL OR @position IS NULL
    BEGIN
        SET @error_msg = 'Invalid data'
        SET @out = 1
        RAISERROR(@error_msg, 16, 1)
        RETURN
    END

    BEGIN TRY
        INSERT INTO Employees VALUES
            (@ID, @first_name, @second_name, @PESEL, @gender, @birth_date, @hire_date, @salary, @phone_number, @address, @department, @position)
    END TRY
    BEGIN CATCH
        SET @error_msg = 'Problem with data'
        SET @out = 1
        RAISERROR(@error_msg, 16, 1)
    END CATCH

RETURN
```

2. Analogicznie do linii

```
CREATE PROCEDURE Add_Line(
 @ID INT = NULL,
 @StartID INT = NULL,
 @EndID INT = NULL,
 @Night BIT = NULL,
 @out BIT OUTPUT
)
AS
DECLARE @error_msg NVARCHAR(100)
    IF @ID is NULL OR @StartID IS NULL OR @EndID IS NULL
    BEGIN
        SET @error_msg = 'Invalid data'
        SET @out = 1
        RAISERROR(@error_msg, 16, 1)
        RETURN
    END

    IF (NOT EXISTS(SELECT * From Stops Where StopID = @StartID))
    BEGIN
        SET @error_msg = 'Invalid data'
        SET @out = 1
        RAISERROR(@error_msg, 16, 1)
        RETURN
    END

    IF (NOT EXISTS(SELECT * From Stops Where StopID = @EndID))
    BEGIN
        SET @error_msg = 'Invalid data'
        SET @out = 1
        RAISERROR(@error_msg, 16, 1)
        RETURN
    END


    BEGIN TRY
        INSERT INTO Lines VALUES
            (@ID, @StartID, @EndID, @Night)
    END TRY
    BEGIN CATCH
        SET @error_msg = 'Problem with data'
        SET @out = 1
        RAISERROR(@error_msg, 16, 1)
    END CATCH

RETURN
```

3. Przyznaje urlop pracownikowi

```
CREATE PROCEDURE Give_Holiday(
 @ID INT = NULL,
 @Start DATE = NULL,
 @End DATE = NULL,
 @out BIT OUTPUT
)
AS
DECLARE @error_msg NVARCHAR(100)

    IF @ID is NULL OR @Start IS NULL OR @End IS NULL
    BEGIN
        SET @error_msg = 'Invalid data'
        SET @out = 1
        RAISERROR(@error_msg, 16, 1)
        RETURN
    END

    IF (NOT EXISTS(SELECT * From Employees Where EmployeeID = @ID))
    BEGIN
        SET @error_msg = 'Invalid data'
        SET @out = 1
        RAISERROR(@error_msg, 16, 1)
        RETURN
    END

    BEGIN TRY
        INSERT INTO Holidays VALUES
            (@ID, @Start, @End)
    END TRY
    BEGIN CATCH

        SET @error_msg = 'Cant give holiday at that time'
        SET @out = 1
        RAISERROR(@error_msg, 16, 1)
    END CATCH

RETURN
```

4. Zwiększa cenę biletów o dany procent

```
CREATE PROCEDURE IncreaceTicketPrice(
 @Change FLOAT = NULL, -- procentowa zmiana stawki za godzine biletu
 @out BIT OUTPUT
)
AS
DECLARE @error_msg NVARCHAR(100)
    IF @change is NULL
    BEGIN
        SET @error_msg = 'Invalid data'
        SET @out = 1
        RAISERROR(@error_msg, 16, 1)
        RETURN
    END

    UPDATE Tickets
    SET Price = Price * @Change


RETURN
```

5. Przyznaje wypłatę pracownikom

```
CREATE PROCEDURE GiveSalary
AS

    DECLARE curs CURSOR FOR SELECT EmployeeID, Salary FROM Employees
    OPEN curs
    DECLARE @ID INT
    DECLARE @Salary MONEY
    DECLARE @Date DATE
    SET @DATE = GETDATE()
    FETCH curs INTO @ID, @Salary

    WHILE (@@FETCH_STATUS = 0)
    BEGIN
        INSERT INTO SalaryHistory VALUES(@ID, @DATE, @Salary)

        FETCH cur INTO @ID, @Salary
    END

    CLOSE curs
    DEALLOCATE curs
```

Pielęgnacja

Dokonywana jest za pomocą polecenia BACK UP DATABASE. Kopię tą trzeba wykonywać regularnie