Systemy Baz Danych

Projekt Hurtownia

2023/2024

Khalima Ismailova

Zakres i krótki opis systemu

Celem projektu jest stworzenie bazy danych dla systemu zarządzania hurtownią. System koncentruje się na zapewnieniu łatwego dostępu do informacji o nabywcach, klientach, zapasach oraz zyskach miesięcznych. Ponadto, umożliwia śledzenie stanu magazynowego, zarządzanie dostawami, zamówieniami oraz rejestrowanie płatności od klientów. Planujemy wykorzystać SQL Server Management Studio (SQL SMS) do implementacji bazy danych.

Wymagania i funkcje systemu

1. Zarządzanie informacjami o przechowywanych produktach:
    - System umożliwia dokładne zarządzanie danymi dotyczącymi zapasów przechowywanych w magazynie, takimi jak identyfikator produkta, nazwa, ilość, ilość minimalna, identyfikator kupującego, cena sprzedaży, cena kupna, czas dostawy
2. Śledzenie stanu magazynowego:
    - Zapasy, których ilość jest mniejsza niż określona minimalna, zostają dodane do tablicy "do_kupienia", co ułatwia utrzymanie odpowiedniego poziomu zapasów w magazynie.
3. Zarządzanie kupującymi i klientami:
    - Przechowywanie danych kupujących takie jak identyfikator kupującego, imię i nazwisko, email, numer telefonu, adres.
    - Przechowywanie danych klientów, w tym identyfikator klienta, imię i nazwisko, email, numer telefonu, adres, historia zamówień.
4. Przetwarzanie płatności, śledzenie nieopłaconych płatności:
    - Obsługa transakcji finansowych, w tym rejestracja płatności, monitorowanie stanu płatności oraz przechowywanie listy klientów, którzy nie uregulowali oczekiwanych płatności.
5. Zarządzanie dostawami:
    - Przechowywanie danych o dostawach, w tym data dostawy, identyfikator zapasu, ilość
6. Przyjmowanie zamówień:
    - Obsługa procesu zamówień, w tym przyjmowanie, przetwarzanie i realizacja zamówień z uwzględnieniem aktualnego stanu magazynowego.
    - Przechowywanie danych o zamówieniach, w tym identyfikator zamówienia, identyfikator klienta, data zamówienia, stan, identyfikatory produktów wraz z ilością i ceną.
7. Śledzenie wydatków, przychodów i zysków dla każdego miesiąca:
    - Automatyczne aktualizowanie tablicy zawierającej przychody i wydatki dla danego miesiąca oraz obliczanie zysku.

Przypadki użycia

Kierownik Hurtowni:

1. **Wprowadzanie nowych produktów:**
    - Dodawanie nowych produktów do bazy danych wraz z informacjami takimi jak identyfikator produktu, identyfikator kupującego, nazwa, opis, ilość, cena sprzedaży, cena kupna, czas dostawy.
2. **Zamawianie dostaw wyczerpujących się zapasów:**
    - Generowanie zamówień na produkty, których ilość w magazynie spadła poniżej określonego poziomu minimalnego.
3. **Zmiana stanu płatności:**
    - Aktualizowanie stanu płatności.
4. **Zmiana stanu zamówienia:**
    - Aktualizowanie stanu zamówienia. Dopuszczalne stany zamówienia to:
        1. Przetwarzane: Zamówienie jest w trakcie realizacji.
        2. Wysłane: Zamówienie zostało wysłane do klienta.
5. **Przegląd klientów z nieuregulowanymi płatnościami:**
    - Przeglądanie listy klientów, którzy nie zapłacili oczekiwanej kwoty, wraz z ich danymi kontaktowymi.

Klient:

1. **Składanie zamówienia:**
    - Składanie zamówień na produkty dostępne w magazynie.
2. **Przegląd historii zamówień:**
    - Przeglądanie historii wcześniej złożonych zamówień, ich stanu.

System:

1. **Aktualizacja dostępności zapasów po ich zamówieniu lub dostarczeniu:**
    - Automatyczne aktualizowanie ilości produktów w magazynie po przyjęciu nowych dostaw lub realizacji zamówień klientów.
2. **Dodawanie zapasów do tabeli "do_kupienia":**
    - Dodawanie produktów do tabeli „do_kupienia” w sytuacji, gdy ich ilość w magazynie spadnie poniżej zdefiniowanego minimalnego poziomu.
3. **Odrzucanie zamówień, gdy nie ma wystarczającej ilości zamawianego produktu:**
    - Automatyczne odrzucanie zamówień w przypadku, gdy ilość zamawianego produktu jest większa niż aktualnie dostępna w magazynie.

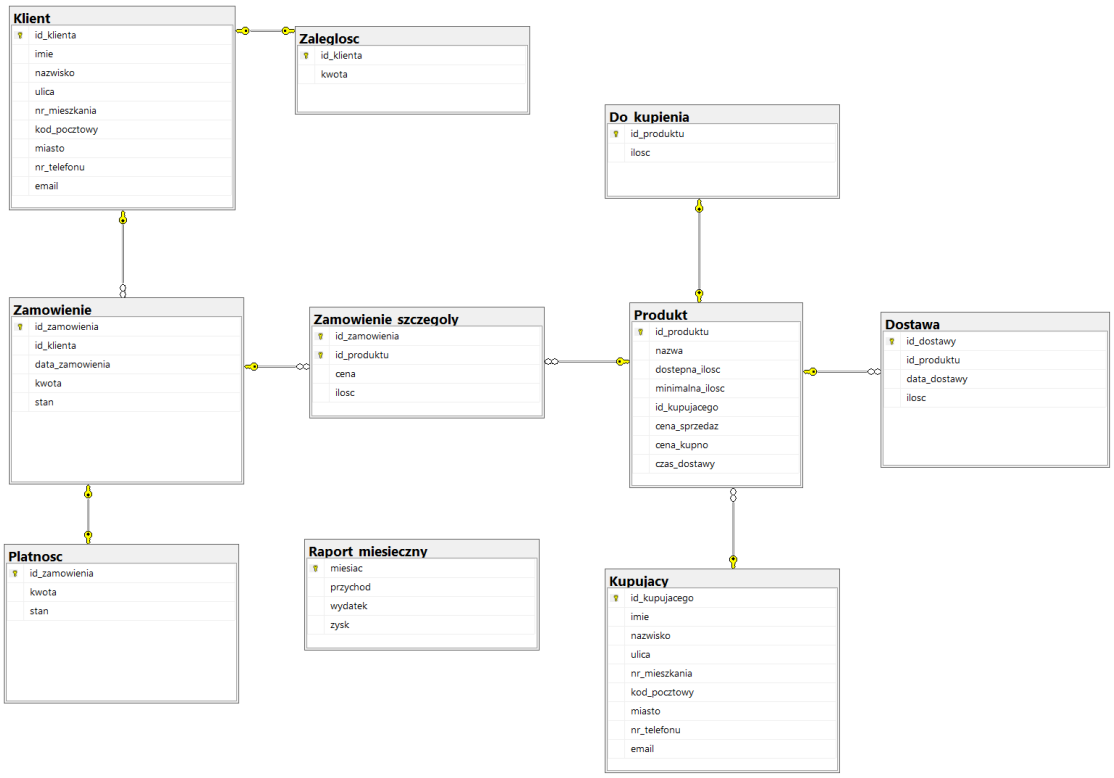
Uproszczenia i założenia

- **Stan płatności:** Pierwotnie kierownik hurtowni będzie ręcznie aktualizował status płatności, jednak w przyszłości system przewiduje integrację z systemem bankowym, aby automatycznie pobierać aktualny stan płatności.
- **Stan zamówienia:** Podobnie, status zamówienia będzie ręcznie aktualizowany przez kierownika hurtowni. W rzeczywistości, informacje o statusie, takie jak

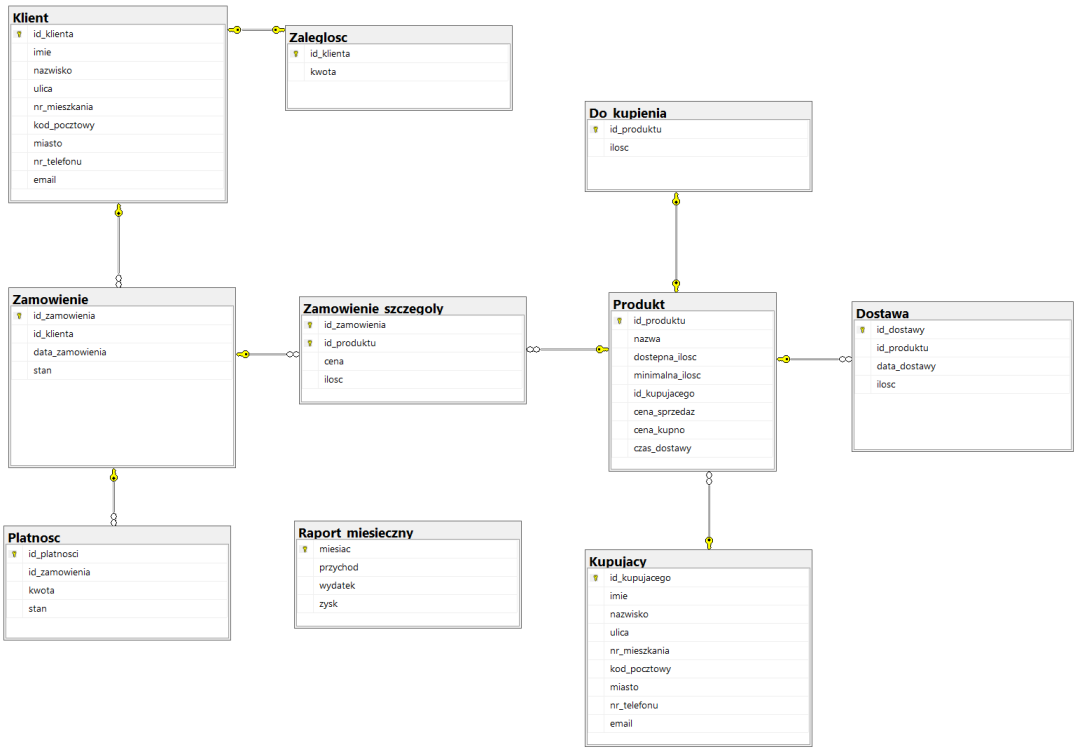
dostawa, mogłyby być automatycznie aktualizowane przez dostawców usług pocztowych.

- **Czas dostawy:** Zakładamy, że dostawy produktów zawsze są realizowane zgodnie z określonym czasem dostawy, jak podano w tabeli (np. data dostawy towaru A jest zawsze tydzień od daty dodania dostawy). Produkt zostaje dodany do zapasów w dzień podany jako data dostawy.

Schemat ER



Pierwsza edycja schematu bazy danych. Później usunięto pole 'kwota' z tabeli 'Zamowienie', aby uniknąć redundancji, ponieważ tabela już zawiera tę informację w relacji z tabelą 'Platnosc'. Dodatkowo, schemat jest znormalizowany do postaci trzeciej postaci normalnej (3NF), co oznacza, że każda kolumna w tabelach zawiera tylko dane jednego rodzaju i jest logicznie powiązana z kluczami głównymi, zapewniając efektywną i spójną strukturę danych.



Ostateczna wersja.

Opis poszczególnych tabel

Produkt		
Nazwa atrybutu	Typ	Opis/Uwagi
id_produktu	int (PK)	Automatycznie inkrementowany identyfikator produktu.
nazwa	nvarchar(64)	Pełna nazwa produktu.
dostepna_ilosc	int	Aktualnie dostępna ilość produktu.
minimalna_ilosc	int	Minimalna dopuszczalna ilość produktu, która powinna być dostępna w magazynie. Jeśli aktualna ilość produktu spada poniżej tej wartości, produkt jest automatycznie dodawany do tabeli „Do_kupienia”
id_kupujacego	int (FK)	Identyfikator kupującego, od którego zamawia się produkt.
cena_sprzedaz	decimal(10, 2)	Cena sprzedaży jednostkowej produktu.
cena_kupno	decimal(10, 2)	Cena zakupu jednostkowego produktu.
czas_dostawy	smallint	Przewidywany czas dostawy produktu w dniach(5 znaczy, że dostawa zajmie 5 dni, 14, że zajmie to 14 dni).

Do_kupienia		
Nazwa atrybutu	Typ	Opis/Uwagi
id_produktu	int (PK, FK)	Automatycznie inkrementowany identyfikator produktu.
ilosc	int	Ilość produktu do zamówienia w celu uzupełnienia zapasów.

Klient		
Nazwa atrybutu	Typ	Opis/Uwagi
id_klienta	int (PK)	Automatycznie inkrementowany identyfikator klienta.
imie	nvarchar(64)	Imię klienta.
nazwisko	nvarchar(100)	Nazwisko klienta.
ulica	nvarchar(112)	Nazwa ulicy miejsca dostawy. Może przyjmować wartości NULL w przypadku osób mieszkających w małych miejscowościach, gdzie nie ma ulic. Przyjęta długość to długość najdłuższej nazwy ulicy w Polsce.
nr_mieszkania	varchar(9)	Numer budynku i mieszkania.
kod_pocztowy	char(6)	Kod pocztowy miejsca dostawy.
miasto	nvarchar(86)	Miejscowość klienta. Przyjęta długość wynika z długości najdłuższej nazwy miejscowości w Polsce.
nr_telefonu	char(9)	Numer telefonu klienta (bez prefiksu kierunkowego, spacji, myślników i innych znaków).
email	varchar(254)	Adres email klienta.

Kupujacy		
Nazwa atrybutu	Typ	Opis/Uwagi
id_kupujacego	int (PK)	Automatycznie inkrementowany identyfikator kupującego.
imie	nvarchar(64)	Imię kupującego.
nazwisko	nvarchar(100)	Nazwisko kupującego.
ulica	nvarchar(112)	Nazwa ulicy miejsca kupującego. Może przyjmować wartości NULL w przypadku osób mieszkających w małych miejscowościach, gdzie nie ma ulic. Przyjęta długość to długość najdłuższej nazwy ulicy w Polsce.
nr_mieszkania	varchar(9)	Numer budynku i mieszkania.
kod_pocztowy	char(6)	Kod pocztowy miejsca kupującego.
miasto	nvarchar(86)	Miejscowość kupującego. Przyjęta długość wynika z długości najdłuższej nazwy miejscowości w Polsce.
nr_telefonu	char(9)	Numer telefonu kupującego (bez prefiksu kierunkowego, spacji, myślników i innych znaków).
email	varchar(254)	Adres email kupującego.

Zamowienie		
Nazwa atrybutu	Typ	Opis/Uwagi
id_zamowienia	int (PK)	Automatycznie inkrementowany identyfikator zamówienia.
id_klienta	int (FK)	Identyfikator klienta, który złożył zamówienie.
data_zamowienia	date	Data złożenia zamówienia.
stan	bit	Stan zamówienia (0 - Przetwarzane, 1 - Wysłane).

Zamowienie_szczegoly		
Nazwa atrybutu	Typ	Opis/Uwagi
id_zamowienia	int (PK, FK)	Identyfikator zamówienia.
id_produktu	int (PK, FK)	Identyfikator produktu, który został zamówiony.
cena	decimal(10, 2)	Cena jednostki produktu.
ilość	int	Ilość zamówionego produktu.

Platnosc		
Nazwa atrybutu	Typ	Opis/Uwagi
id_platnosci	int (PK)	Automatycznie inkrementowany identyfikator płatności.
id_zamowienia	int (FK)	Identyfikator zamówienia, do którego należy płatność.
kwota	decimal(10, 2)	Suma płatności za zamówienie.
stan	bit	Stan płatności (1 - Wpłacona, 0 - Oczekuje na rozpatrzenie).

Zaleglosc		
Nazwa atrybutu	Typ	Opis/Uwagi
id_klienta	int (PK, FK)	Identyfikator klienta z zaległościami płatniczymi.
kwota	decimal(10, 2)	Kwota zaległości do zapłacenia. Gdy osiąga 0, wpis zostaje usunięty z tabeli.

Dostawa		
Nazwa atrybutu	Typ	Opis/Uwagi
id_dostawy	int (PK)	Automatycznie inkrementowany identyfikator dostawy.
id_produktu	int(FK)	Identyfikator produktu dostarczanego.
data_dostawy	date	Data realizacji dostawy.
ilosc	int	Ilość dostarczanego produktu.

Raport_miesieczny		
Nazwa atrybutu	Typ	Opis/Uwagi
miesiac	date (PK)	Data miesiąca, dla którego jest tworzony raport.
przychod	decimal(15, 2)	Suma przychodów w danym miesiącu.
wydatek	decimal(15, 2)	Suma wydatków w danym miesiącu.
zysk	decimal(15, 2)	Zysk wygenerowany w danym miesiącu (przychód minus wydatek).

Implementacja tablic

```
use ismailov

-- Klient
CREATE TABLE Klient (
    id_klienta INT IDENTITY(1,1) PRIMARY KEY,
    imie NVARCHAR(64) NOT NULL,
    nazwisko NVARCHAR(100) NOT NULL,
    ulica NVARCHAR(112) NULL,
    nr_mieszkania VARCHAR(9) NOT NULL,
    kod_pocztowy CHAR(6) NOT NULL,
    miasto NVARCHAR(86) NOT NULL,
    nr_telefonu CHAR(9) NOT NULL,
    email VARCHAR(254) NOT NULL
);
-- Zaleglosc
CREATE TABLE Zaleglosc (
    id_klienta INT PRIMARY KEY,
    kwota DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (id_klienta) REFERENCES Klient (id_klienta)
);
-- Kupujacy
CREATE TABLE Kupujacy (
    id_kupujacego INT IDENTITY(1,1) PRIMARY KEY,
    imie NVARCHAR(64) NOT NULL,
    nazwisko NVARCHAR(100) NOT NULL,
    ulica NVARCHAR(112) NULL,
    nr_mieszkania VARCHAR(9) NOT NULL,
    kod_pocztowy CHAR(6) NOT NULL,
    miasto NVARCHAR(86) NOT NULL,
    nr_telefonu CHAR(9) NOT NULL,
    email VARCHAR(254) NOT NULL
);
-- Produkt
CREATE TABLE Produkt (
    id_produktu INT IDENTITY(1,1) PRIMARY KEY,
    nazwa NVARCHAR(64) NOT NULL,
    dostepna_ilosc INT NOT NULL,
    minimalna_ilosc INT NOT NULL,
    id_kupujacego INT NOT NULL,
    cena_sprzedaz DECIMAL(10, 2) NOT NULL,
    cena_kupno DECIMAL(10, 2) NOT NULL,
```

```
    czas_dostawy SMALLINT NOT NULL,
    FOREIGN KEY (id_kupujacego) REFERENCES Kupujacy (id_kupujacego)
);
-- Do_kupienia
CREATE TABLE Do_kupienia (
    id_produktu INT PRIMARY KEY,
    ilosc INT NOT NULL,
    FOREIGN KEY (id_produktu) REFERENCES Produkt (id_produktu)
);
-- Dostawa
CREATE TABLE Dostawa (
    id_dostawy INT IDENTITY(1,1) PRIMARY KEY,
    id_produktu INT NOT NULL,
    data_dostawy DATE NOT NULL,
    ilosc INT NOT NULL,
    FOREIGN KEY (id_produktu) REFERENCES Produkt (id_produktu)
);
-- Zamowienie
CREATE TABLE Zamowienie (
    id_zamowienia INT IDENTITY(1,1) PRIMARY KEY,
    id_klienta INT NOT NULL,
    data_zamowienia DATE NOT NULL,
    stan BIT NOT NULL,
       FOREIGN KEY (id_klienta) REFERENCES Klient (id_klienta)
);
-- Zamowienie_szczegoly
CREATE TABLE Zamowienie_szczegoly (
    id_zamowienia INT NOT NULL,
    id_produktu INT NOT NULL,
    cena DECIMAL(10, 2) NOT NULL,
    ilosc INT NOT NULL,
       PRIMARY KEY CLUSTERED (id_zamowienia, id_produktu),
    FOREIGN KEY (id_zamowienia) REFERENCES Zamowienie (id_zamowienia),
    FOREIGN KEY (id_produktu) REFERENCES Produkt (id_produktu)
);
-- Platnosc
CREATE TABLE Platnosc (
    id_platnosci INT IDENTITY(1,1) PRIMARY KEY,
    id_zamowienia INT NOT NULL,
    kwota DECIMAL(10, 2) NOT NULL,
    stan BIT NOT NULL,
    FOREIGN KEY (id_zamowienia) REFERENCES Zamowienie (id_zamowienia)
);
-- Raport_miesieczny
CREATE TABLE Raport_miesieczny (
    miesiac DATE PRIMARY KEY,
    przychod DECIMAL(15, 2) NOT NULL,
    wydatek DECIMAL(15, 2) NOT NULL,
    zysk DECIMAL(15, 2) NOT NULL
);
```

Procedury, triggery

Procedura `p_Zloz_zamowienie` umożliwia klientowi złożenie zamówienia, przyjmując identyfikator klienta i tabelę produktów. Tworzy nowe zamówienie w tabeli `Zamowienie`, weryfikuje dostępność produktów, dodaje szczegóły do `Zamowienie_szczegoly`, zmniejsza ilość produktów w magazynie, oblicza kwotę zamówienia, aktualizuje raport w `Raport_miesieczny`, tworzy płatność w `Platnosc`, oraz zarządza zaległościami klienta w tabeli `Zaleglosc`. W przypadku błędów, procedura cofa transakcję i resetuje identyfikator zamówienia.

```
CREATE TYPE ProduktyTable AS TABLE (
    id_produktu INT,
    ilosc INT
);

CREATE PROCEDURE p_Zloz_zamowienie
    @id_klienta INT,
    @produkty ProduktyTable READONLY
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @order_id INT;
    BEGIN TRY
        BEGIN TRANSACTION;


        INSERT INTO Zamowienie (id_klienta, data_zamowienia, stan)
        VALUES (@id_klienta, GETDATE(), 0);

        SET @order_id = SCOPE_IDENTITY();

        DECLARE @za_malo BIT = 0;
              -- sprawdzamy, czy produkty sa dostepne
        INSERT INTO Zamowienie_szczegoly (id_zamowienia, id_produktu, cena, ilosc)
        SELECT @order_id, p.id_produktu, p.cena_sprzedaz, op.ilosc
        FROM @produkty op
        JOIN Produkt p ON op.id_produktu = p.id_produktu
        WHERE p.dostepna_ilosc >= op.ilosc;

        -- czy wszystkie produkty zostaly dodane do zamowienia
        SET @za_malo = CASE WHEN EXISTS (
            SELECT 1
            FROM @produkty op
            LEFT JOIN Produkt p ON op.id_produktu = p.id_produktu
            WHERE p.id_produktu IS NULL OR p.dostepna_ilosc < op.ilosc
        ) THEN 1 ELSE 0 END;

        IF @za_malo = 1
        BEGIN
            THROW 50001, 'Nie wystarczające zapasy na zamówienie', 1;
        END;

        -- zmniejszamy zapasy w tabeli Produkt
        UPDATE p
        SET dostepna_ilosc = dostepna_ilosc - op.ilosc
        FROM @produkty op
        JOIN Produkt p ON op.id_produktu = p.id_produktu;

        -- aktualizujemy przychody
        DECLARE @kwota DECIMAL(10, 2);
```

```
        SELECT @kwota = SUM(op.ilosc * p.cena_sprzedaz)
        FROM @produkty op
        JOIN Produkt p ON op.id_produktu = p.id_produktu;


        DECLARE @current_month DATE = DATEFROMPARTS(YEAR(GETDATE()), MONTH(GETDATE()),
1);

        MERGE INTO Raport_miesieczny AS target
        USING (
            SELECT
                @current_month AS miesiac,
                @kwota AS przychod,
                0 AS wydatek,
                @kwota AS zysk
        ) AS source (miesiac, przychod, wydatek, zysk)
        ON target.miesiac = source.miesiac
        WHEN MATCHED THEN
            UPDATE SET
                target.przychod = target.przychod + source.przychod,
                target.zysk = target.zysk + source.zysk
        WHEN NOT MATCHED THEN
            INSERT (miesiac, przychod, wydatek, zysk)
            VALUES (source.miesiac, source.przychod, source.wydatek, source.zysk);

        -- tworzymy platnosc
        INSERT INTO Platnosc (id_zamowienia, kwota, stan)
        VALUES (@order_id, @kwota, 0);

        -- zarzadzamy zaleglosciami
              -- czy klient ma już zaleglosci
              IF EXISTS (
                     SELECT 1
                     FROM Zaleglosc
                     WHERE id_klienta = @id_klienta
              )
              BEGIN
                     -- aktualizacja istniejącej zaległości
                     UPDATE Zaleglosc
                     SET kwota = kwota + @kwota
                     WHERE id_klienta = @id_klienta;
              END
              ELSE
              BEGIN
                     -- dodanie nowej zaległości
                     INSERT INTO Zaleglosc (id_klienta, kwota)
                     VALUES (@id_klienta, @kwota);
              END


        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
              ROLLBACK TRANSACTION;
              DECLARE @last_id INT;
        SELECT @last_id = ISNULL(MAX(id_zamowienia), 0) FROM Zamowienie;
              DBCC CHECKIDENT ('Zamowienie', RESEED, @last_id);

        THROW;
    END CATCH;
END;
```

Procedura `p_Utworz_dostawe` służy do tworzenia dostawy dla produktu znajdującego się na liście `do_kupienia`. Procedura przyjmuje identyfikator produktu jako parametr wejściowy. Najpierw sprawdza, czy produkt jest na liście `do_kupienia`, a następnie pobiera ilość oraz czas dostawy dla tego produktu. Na podstawie tych danych oblicza datę dostawy i całkowity wydatek związany z zakupem produktu. Procedura aktualizuje miesięczny raport finansowy w tabeli `Raport_miesieczny`, dodając wydatki związane z zakupem, a następnie dodaje wpis o nowej dostawie w tabeli `Dostawa`. Po zakończeniu operacji, produkt jest usuwany z tabeli `do_kupienia`, aby uniknąć redundancji.

```
CREATE PROCEDURE p_Utworz_dostawe
    @id_produktu INT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @ilosc INT;
    DECLARE @czas_dostawy INT;
    DECLARE @data_dostawy DATE;
       DECLARE @wydatek DECIMAL(10, 2) = 0;

    -- czy produkt jest w tabeli do_kupienia
    IF NOT EXISTS (
        SELECT 1
        FROM do_kupienia
        WHERE id_produktu = @id_produktu
    )
    BEGIN
        THROW 50001, 'Produkt nie istnieje na liście do kupienia.', 1;
        RETURN;
    END

    SELECT @ilosc = ilosc
    FROM do_kupienia
    WHERE id_produktu = @id_produktu;

       SELECT @czas_dostawy = czas_dostawy
    FROM Produkt
    WHERE id_produktu = @id_produktu;

        -- liczymy date dostawy
    SET @data_dostawy = DATEADD(DAY, @czas_dostawy, GETDATE());

       -- aktualizowanie raportu miesiecznego
    DECLARE @current_month DATE = DATEFROMPARTS(YEAR(GETDATE()), MONTH(GETDATE()), 1);

       SELECT @wydatek = @ilosc * cena_kupno
    FROM Produkt
    WHERE id_produktu = @id_produktu;

    MERGE INTO Raport_miesieczny AS target
    USING (
        SELECT
            @current_month AS miesiac,
            0 AS przychod,
            @wydatek AS wydatek,
                    -@wydatek AS zysk
    ) AS source (miesiac, przychod, wydatek, zysk)
    ON target.miesiac = source.miesiac
```

```
    WHEN MATCHED THEN
        UPDATE SET
            target.wydatek = target.wydatek + source.wydatek,
            target.zysk = target.zysk + source.zysk
    WHEN NOT MATCHED THEN
        INSERT (miesiac, przychod, wydatek, zysk)
        VALUES (source.miesiac, source.przychod, source.wydatek, source.zysk);

    INSERT INTO Dostawa (id_produktu, ilosc, data_dostawy)
    VALUES (@id_produktu, @ilosc, @data_dostawy);

    DELETE FROM do_kupienia
    WHERE id_produktu = @id_produktu;

    PRINT 'Dostawa utworzona pomyślnie.';
END;
```

Trigger `tr_Sprawdz_ilosc` uruchamiany po aktualizacji tabeli `Produkt`, sprawdza, czy ilość dostępna produktu jest mniejsza od minimalnej. Jeśli tak, oblicza potrzebną ilość (trzykrotność minimalnej) i dodaje wpis do tabeli `do_kupienia`, chyba że taki wpis już istnieje. Trigger używa kursora do przetwarzania każdego zaktualizowanego wiersza, zapewniając, że zapotrzebowanie na produkty jest monitorowane i odpowiednio rejestrowane.

```
CREATE TRIGGER tr_Sprawdz_ilosc
ON Produkt
AFTER UPDATE
AS
BEGIN
    DECLARE @id_produktu INT;
    DECLARE @nowa_ilosc INT;
    DECLARE @minimalna_ilosc INT;
    DECLARE @potrzebna_ilosc INT;

       DECLARE cur CURSOR FOR
    SELECT i.id_produktu, i.dostepna_ilosc, i.minimalna_ilosc
    FROM inserted i
    INNER JOIN Produkt p ON i.id_produktu = p.id_produktu;
         OPEN cur;
    FETCH NEXT FROM cur INTO @id_produktu, @nowa_ilosc, @minimalna_ilosc;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF @nowa_ilosc < @minimalna_ilosc
        BEGIN
             -- obliczamy potrzebna ilosc do uzupelnienia
            SET @potrzebna_ilosc = 3 * @minimalna_ilosc;

            -- sprawdzamy czy taki rekord juz istnieje
            IF NOT EXISTS (
                SELECT 1
                FROM do_kupienia
                WHERE id_produktu = @id_produktu
            )
            BEGIN

                INSERT INTO do_kupienia (id_produktu, ilosc)
                VALUES (@id_produktu, @potrzebna_ilosc);
            END
        END

        FETCH NEXT FROM cur INTO @id_produktu, @nowa_ilosc, @minimalna_ilosc;
    END

    CLOSE cur;
    DEALLOCATE cur;
END;
```

Procedura `p_Historia_zamowien` służy do wyświetlania historii zamówień dla określonego klienta. Przyjmuje identyfikator klienta jako parametr wejściowy i zwraca szczegóły zamówień dokonanych przez tego klienta. Dane obejmują identyfikator zamówienia, datę zamówienia, stan zamówienia oraz kwotę płatności za zamówienie. Wyniki są posortowane malejąco według daty zamówienia, co oznacza, że najnowsze zamówienia będą wyświetlane jako pierwsze.

```
CREATE PROCEDURE p_Historia_zamowien
    @id_klienta INT
AS
BEGIN
   SET NOCOUNT ON;

    SELECT
        z.id_zamowienia AS 'ID Zamówienia',
        z.data_zamowienia AS 'Data Zamówienia',
        z.stan AS 'Stan Zamówienia',
        p.kwota AS 'Kwota'
    FROM Zamowienie z
    JOIN Platnosc p ON z.id_zamowienia = p.id_zamowienia
    WHERE z.id_klienta = @id_klienta
    ORDER BY z.data_zamowienia DESC;


END;
```

Procedura `p_Przetworz_Platnosc` przetwarza płatność klienta. Aktualizuje status płatności na przetworzony, zmniejsza zadłużenie klienta o kwotę płatności i usuwa klienta z tabeli `Zaleglosc`, jeśli jego zadłużenie spadnie poniżej lub równo zero.

```
CREATE PROCEDURE p_Przetworz_Platnosc
    @id_platnosci INT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @kwota DECIMAL(10, 2);
    DECLARE @id_klienta INT;

    BEGIN TRY
        BEGIN TRANSACTION;

             IF EXISTS (SELECT 1 FROM Platnosc WHERE id_platnosci = @id_platnosci AND
stan = 1)
        BEGIN
            --  jest juz przetworzona, nic nie robimy
            RETURN;
        END;
        -- aktualizacja stanu platnosci
        UPDATE Platnosc
        SET stan = 1
        WHERE id_platnosci = @id_platnosci;

        -- pobranie kwoty platnosci
        SELECT @kwota = kwota
        FROM Platnosc
        WHERE id_platnosci = @id_platnosci;

        -- zmniejszenie zadluzenia
        SELECT @id_klienta = id_klienta
        FROM Zamowienie
        WHERE id_zamowienia = (SELECT id_zamowienia FROM Platnosc WHERE id_platnosci =
@id_platnosci);

        UPDATE Zaleglosc
        SET kwota = kwota - @kwota
        WHERE id_klienta = @id_klienta;

        -- Usunięcie klienta z tabeli Zaleglosc, jeśli zadłużenie <= 0
        DELETE FROM Zaleglosc
        WHERE id_klienta = @id_klienta AND kwota <= 0;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        THROW;
    END CATCH;
END;
```

Wprowadzenie danych przykładowych

Na początku wypełniamy tablicy `Klient`, `Kupujacy`, `Produkt`, oraz `Raport_miesieczny` danymi:

```
INSERT INTO Klient (imie, nazwisko, ulica, nr_mieszkania, kod_pocztowy, miasto,
nr_telefonu, email) VALUES
('Jan', 'Kowalski', 'Kwiatowa', '15/1A', '00-123', 'Warszawa', '123456789',
'jan.kowalski@example.com'),
('Anna', 'Nowak', 'Słoneczna', '20/4B', '01-234', 'Kraków', '234567890',
'anna.nowak@example.com'),
('Piotr', 'Wiśniewski', 'Lipowa', '10/3C', '02-345', 'Łódź', '345678901',
'piotr.wisniewski@example.com'),
('Katarzyna', 'Wójcik', 'Brzozowa', '8/5D', '03-456', 'Wrocław', '456789012',
'katarzyna.wojcik@example.com'),
('Michał', 'Kamiński', 'Dębowa', '12/6E', '04-567', 'Poznań', '567890123',
'michal.kaminski@example.com'),
('Agnieszka', 'Lewandowska', 'Topolowa', '3/7F', '05-678', 'Gdańsk', '678901234',
'agnieszka.lewandowska@example.com'),
('Tomasz', 'Zieliński', 'Jesionowa', '22/8G', '06-789', 'Szczecin', '789012345',
'tomasz.zielinski@example.com'),
('Magdalena', 'Szymańska', 'Wierzbowo', '19/9H', '07-890', 'Bydgoszcz', '890123456',
'magdalena.szymanska@example.com'),
('Robert', 'Woźniak', 'Grabowa', '11/10I', '08-901', 'Lublin', '901234567',
'robert.wozniak@example.com'),
('Elżbieta', 'Dąbrowska', 'Sosnowa', '25/11J', '09-012', 'Katowice', '012345678',
'elzbieta.dabrowska@example.com');
```

```
INSERT INTO Kupujacy (imie, nazwisko, ulica, nr_mieszkania, kod_pocztowy, miasto,
nr_telefonu, email) VALUES
('Grzegorz', 'Mazur', 'Wiejska', '8', '66-777', 'Gdynia', '101202303',
'grzegorz.mazur@example.com'),
('Ewa', 'Kaczmarek', NULL, '32', '77-888', 'Rzeszów', '202303404',
'ewa.kaczmarek@example.com'),
('Łukasz', 'Król', 'Warszawska', '12B/4', '88-999', 'Białystok', '303404505',
'lukasz.krol@example.com'),
('Magdalena', 'Pawlak', 'Ogrodowa', '6/1', '99-000', 'Bydgoszcz', '404505606',
'magdalena.pawlak@example.com'),
('Rafał', 'Włodarczyk', 'Świętokrzyska', '14', '11-222', 'Katowice', '505606707',
'rafal.wlodarczyk@example.com'),
('Alicja', 'Dąbrowska', NULL, '27', '22-333', 'Sopot', '606707808',
'alicja.dabrowska@example.com'),
('Krystyna', 'Górska', 'Krucza', '9/3', '33-444', 'Gliwice', '707808909',
'krystyna.gorska@example.com'),
('Dariusz', 'Zając', 'Lipowa', '1A', '44-555', 'Toruń', '808909010',
'dariusz.zajac@example.com'),
('Natalia', 'Błaszczyk', 'Bukowa', '19B/7', '55-666', 'Opole', '909010111',
'natalia.blaszczyk@example.com'),
('Marek', 'Chmielewski', 'Cicha', '4', '66-777', 'Radom', '010111212',
'marek.chmielewski@example.com');
```

```
INSERT INTO Produkt (nazwa, dostepna_ilosc, minimalna_ilosc, id_kupujacego,
cena_sprzedaz, cena_kupno, czas_dostawy) VALUES
('Długopis', 100, 20, 1, 1.50, 0.80, 3),
('Ołówek HB', 150, 30, 2, 0.90, 0.50, 2),
```

```
('Notes A4', 75, 10, 3, 12.00, 7.50, 5),
('Kalendarz 2024', 50, 5, 4, 25.00, 15.00, 7),
('Blok rysunkowy', 200, 50, 5, 8.00, 5.00, 4),
('Markery kolorowe', 120, 25, 6, 10.00, 6.00, 3),
('Zeszyt 60 kartek', 300, 60, 7, 3.50, 2.00, 2),
('Taśma klejąca', 180, 40, 8, 4.00, 2.50, 1),
('Klej biurowy', 90, 15, 9, 5.00, 3.00, 3),
('Korektor w taśmie', 110, 20, 10, 7.50, 4.50, 2),
('Linijka 30 cm', 140, 25, 1, 2.50, 1.50, 1),
('Nożyczki biurowe', 130, 30, 2, 6.00, 3.50, 2),
('Zakreślacze', 160, 35, 3, 9.00, 5.50, 4),
('Segregator A4', 20, 15, 4, 15.00, 10.00, 5),
('Etykiety samoprzylepne', 170, 30, 5, 5.50, 3.00, 2),
('Papier ksero', 250, 50, 6, 20.00, 12.00, 3),
('Teczka papierowa', 140, 25, 7, 8.50, 5.00, 2),
('Kalkulator biurowy', 60, 10, 8, 35.00, 20.00, 7),
('Zszywacz', 110, 20, 9, 14.00, 8.00, 3),
('Zszywki do zszywacza', 190, 40, 10, 3.00, 1.50, 1),
('Spinacze biurowe', 220, 50, 1, 2.00, 1.00, 1),
('Pióro wieczne', 70, 10, 2, 45.00, 30.00, 6),
('Kreda kolorowa', 150, 30, 3, 6.50, 3.50, 2),
('Farby akwarelowe', 95, 15, 4, 18.00, 10.00, 5),
('Pędzle do farb', 125, 20, 5, 12.00, 7.00, 3);
```

```
INSERT INTO Raport_miesieczny (miesiac, przychod, wydatek, zysk) VALUES
('2024-01-01', 50000.00, 40000.00, 10000.00),
('2024-02-01', 55000.00, 45000.00, 10000.00),
('2024-03-01', 60000.00, 48000.00, 12000.00),
('2024-04-01', 65000.00, 50000.00, 15000.00),
('2024-05-01', 70000.00, 52000.00, 18000.00);
```

Po zainicjowaniu danych, pozostałe operacje w systemie są realizowane za pomocą procedur. Składamy zamówienia:

```
--  1
DECLARE @produkty1 ProduktyTable;
DELETE FROM @produkty1;
INSERT INTO @produkty1 (id_produktu, ilosc)
VALUES (1, 10), (2, 5), (3, 3);
EXEC p_Zloz_zamowienie 1, @produkty1;

-- 2
DECLARE @produkty2 ProduktyTable;
DELETE FROM @produkty2;
INSERT INTO @produkty2 (id_produktu, ilosc)
VALUES (2, 8), (4, 4), (6, 5), (25,10);
EXEC p_Zloz_zamowienie 2, @produkty2;

-- 3
DECLARE @produkty3 ProduktyTable;
DELETE FROM @produkty3;
INSERT INTO @produkty3 (id_produktu, ilosc)
VALUES (3, 5), (5, 10), (7, 7);
EXEC p_Zloz_zamowienie 3, @produkty3;

DECLARE @produkty32 ProduktyTable;
DELETE FROM @produkty32;
INSERT INTO @produkty32 (id_produktu, ilosc)
```

```
VALUES (11, 30), (13, 40), (25, 100);
EXEC p_Zloz_zamowienie 3, @produkty32;

-- 4
DECLARE @produkty4 ProduktyTable;
DELETE FROM @produkty4;
INSERT INTO @produkty4 (id_produktu, ilosc)
VALUES (4, 2), (6, 3), (8, 4), (20, 40);
EXEC p_Zloz_zamowienie 4, @produkty4;

-- 5
DECLARE @produkty5 ProduktyTable;
DELETE FROM @produkty5;
INSERT INTO @produkty5 (id_produktu, ilosc)
VALUES (5, 15), (7, 8);
EXEC p_Zloz_zamowienie 5, @produkty5;

-- 6
DECLARE @produkty6 ProduktyTable;
DELETE FROM @produkty6;
INSERT INTO @produkty6 (id_produktu, ilosc)
VALUES (6, 6), (7, 250), (9, 6), (10, 4);
EXEC p_Zloz_zamowienie 6, @produkty6;

-- 7
DECLARE @produkty7 ProduktyTable;
DELETE FROM @produkty7;
INSERT INTO @produkty7 (id_produktu, ilosc)
VALUES (7, 9), (9, 5);
EXEC p_Zloz_zamowienie 7, @produkty7;

-- 8
DECLARE @produkty8 ProduktyTable;
DELETE FROM @produkty8;
INSERT INTO @produkty8 (id_produktu, ilosc)
VALUES (8, 4), (10, 2), (12, 3);
EXEC p_Zloz_zamowienie 8, @produkty8;

-- 9
DECLARE @produkty9 ProduktyTable;
DELETE FROM @produkty9;
INSERT INTO @produkty9 (id_produktu, ilosc)
VALUES (9, 7), (11, 5), (13, 4);
EXEC p_Zloz_zamowienie 9, @produkty9;

-- 10
DECLARE @produkty10 ProduktyTable;
DELETE FROM @produkty10;
INSERT INTO @produkty10 (id_produktu, ilosc)
VALUES (10, 3), (12, 2), (14, 4);
EXEC p_Zloz_zamowienie 10, @produkty10;


DECLARE @produkty102 ProduktyTable;
DELETE FROM @produkty102;
INSERT INTO @produkty102 (id_produktu, ilosc)
VALUES (11, 30), (13, 40), (21, 200);
EXEC p_Zloz_zamowienie 10, @produkty102;
```

Efekty złożenia zamówień:

Tabela `Zamowienie`

	id_zamowienia	id_klienta	data_zamowienia	stan
1	1	1	2024-06-28	0
2	2	2	2024-06-28	0
3	3	3	2024-06-28	0
4	4	3	2024-06-28	0
5	5	4	2024-06-28	0
6	6	5	2024-06-28	0
7	7	6	2024-06-28	0
8	8	7	2024-06-28	0
9	9	8	2024-06-28	0
10	10	9	2024-06-28	0
11	11	10	2024-06-28	0
12	12	10	2024-06-28	0

Tabela `Platnosci`

	id_platnosci	id_zamowienia	kwota	stan
1	1	1	55.50	0
2	2	2	277.20	0
3	3	3	164.50	0
4	4	4	1635.00	0
5	5	5	216.00	0
6	6	6	148.00	0
7	7	7	995.00	0
8	8	8	56.50	0
9	9	9	49.00	0
10	10	10	83.50	0
11	11	11	94.50	0
12	12	12	835.00	0

Tabela `Zamowienie_szczegoly`

	id_zamowienia	id_produktu	cena	ilosc
1	1	1	1.50	10
2	1	2	0.90	5
3	1	3	12.00	3
4	2	2	0.90	8
5	2	4	25.00	4
6	2	6	10.00	5
7	2	25	12.00	10
8	3	3	12.00	5
9	3	5	8.00	10
10	3	7	3.50	7
11	4	11	2.50	30
12	4	13	9.00	40
13	4	25	12.00	100
14	5	4	25.00	2
15	5	6	10.00	3
16	5	8	4.00	4
17	5	20	3.00	40
18	6	5	8.00	15
19	6	7	3.50	8
20	7	6	10.00	6
21	7	7	3.50	250
22	7	9	5.00	6
23	7	10	7.50	4

Tabela `Zaleglosc`

	id_klienta	kwota
1	1	55.50
2	2	277.20
3	3	1799.50
4	4	216.00
5	5	148.00
6	6	995.00
7	7	56.50
8	8	49.00
9	9	83.50
10	10	929.50

Tabela `Raport_miesieczny`

	miesiac	przychod	wydatek	zysk
1	2024-01-01	50000.00	40000.00	10000.00
2	2024-02-01	55000.00	45000.00	10000.00
3	2024-03-01	60000.00	48000.00	12000.00
4	2024-04-01	65000.00	50000.00	15000.00
5	2024-05-01	70000.00	52000.00	18000.00
6	2024-06-01	4609.70	0.00	4609.70

Zadziała trigger `t_Sprawdz_ilosc` i doda produkty do tabeli `Do_kupienia`

	id_produktu	ilosc
1	7	180
2	21	150
3	25	60

Przetwarzamy część płatności:

```
EXEC p_przetworz_platnosc 1;
EXEC p_przetworz_platnosc 2;
EXEC p_przetworz_platnosc 3;
EXEC p_przetworz_platnosc 4;
EXEC p_przetworz_platnosc 5;
EXEC p_przetworz_platnosc 6;
EXEC p_przetworz_platnosc 10;
```

W `Platnosci` stany są zmieniane na 1:

	id_platnosci	id_zamowienia	kwota	stan
1	1	1	55.50	1
2	2	2	277.20	1
3	3	3	164.50	1
4	4	4	1635.00	1
5	5	5	216.00	1
6	6	6	148.00	1
7	7	7	995.00	0
8	8	8	56.50	0
9	9	9	49.00	0
10	10	10	83.50	1
11	11	11	94.50	0
12	12	12	835.00	0

Tez zostaje aktualizowana tabela `Zaleglosci`:

	id_klienta	kwota
1	6	995.00
2	7	56.50
3	8	49.00
4	10	929.50

Teraz możemy utworzyć dostawę kończących się zapasów:

	id_produktu	ilosc
1	7	180
2	21	150
3	25	60

```
EXEC p_Utworz_dostawe 21;
EXEC p_Utworz_dostawe 25;
```

Znikają one z tabeli `Do_kupienia`:

	id_produktu	ilosc
1	7	180

I już mamy wpisy w `Dostawa`:

	id_dostawy	id_produktu	data_dostawy	ilosc
1	5	21	2024-06-29	150
2	6	25	2024-07-01	60

Oraz zaktualizowaną tabelę `Raport_miesieczny`:

	miesiac	przychod	wydatek	zysk
1	2024-01-01	50000.00	40000.00	10000.00
2	2024-02-01	55000.00	45000.00	10000.00
3	2024-03-01	60000.00	48000.00	12000.00
4	2024-04-01	65000.00	50000.00	15000.00
5	2024-05-01	70000.00	52000.00	18000.00
6	2024-06-01	4609.70	570.00	4039.70

Na koniec możemy sprawdzić historie zamówień jednego z klientów:

```
EXEC p_Historia_zamowien 10;
```

	ID Zamówienia	Data Zamówienia	Stan Zamówienia	Kwota
1	11	2024-06-28	0	94.50
2	12	2024-06-28	0	835.00