Projekt bazy danych do zarządzania produkcją i sprzedażą w firmie meblarskiej

Adam Sokołowski
Wiktor Stokłosa
Wojciech Bernacki

Podstawy Baz Danych 2025/2026, grupa 12

Spis treści

1. Wstęp

Projekt został zrealizowany w ramach przedmiotu **Podstawy Baz Danych (PBD)** na kierunku *Informatyka* na Wydziale Informatyki Akademii Górniczo-Hutniczej w roku akademickim **2025/2026**.

Celem projektu było zaprojektowanie oraz implementacja relacyjnej bazy danych wspierającej funkcjonowanie firmy produkcyjno-sprzedażowej. Opracowany system obejmuje pełny cykl działalności przedsiębiorstwa – od ewidencji klientów i pracowników, poprzez obsługę zamówień, planowanie i realizację produkcji, zarządzanie magazynem komponentów i produktów, aż po fakturowanie, płatności, dostawy oraz obsługę reklamacji.

Zaprojektowana baza danych składa się z powiązanych modułów funkcjonalnych, w tym m.in.:

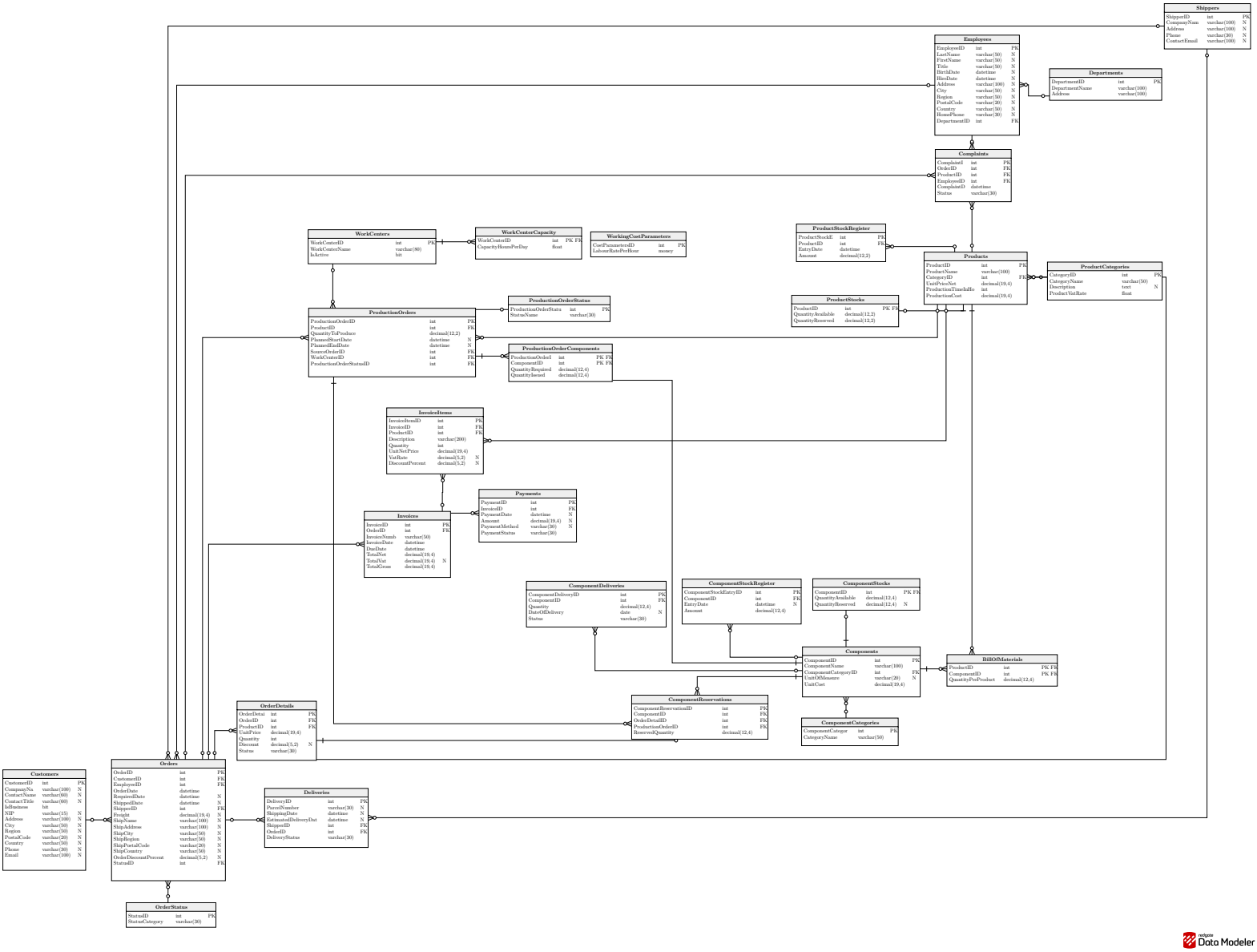
- **sprzedaży i zamówień** (Orders, OrderDetails, Customers),
- **produkcji** (ProductionOrders, ProductionOrderComponents, WorkCenters, ProductionOrderStatus),
- **magazynu komponentów i produktów** (Components, ComponentStocks, ProductStocks, rejestry magazynowe),
- **rozliczeń finansowych** (Invoices, InvoiceItems, Payments),
- **logistyki i dostaw** (Deliveries, Shippers),
- **struktury organizacyjnej firmy** (Employees, Departments).

System został zaimplementowany w środowisku **Microsoft SQL Server**, a model bazy danych opracowano przy użyciu narzędzia **Redgate Data Modeler**. Projekt spełnia założenia normalizacji danych oraz wykorzystuje mechanizmy integralności referencyjnej, ograniczenia CHECK, klucze główne i obce, a także dodatkowe obiekty bazy danych, takie jak **widoki raportowe, procedury składowane, funkcje, triggery, indeksy oraz role użytkowników**.

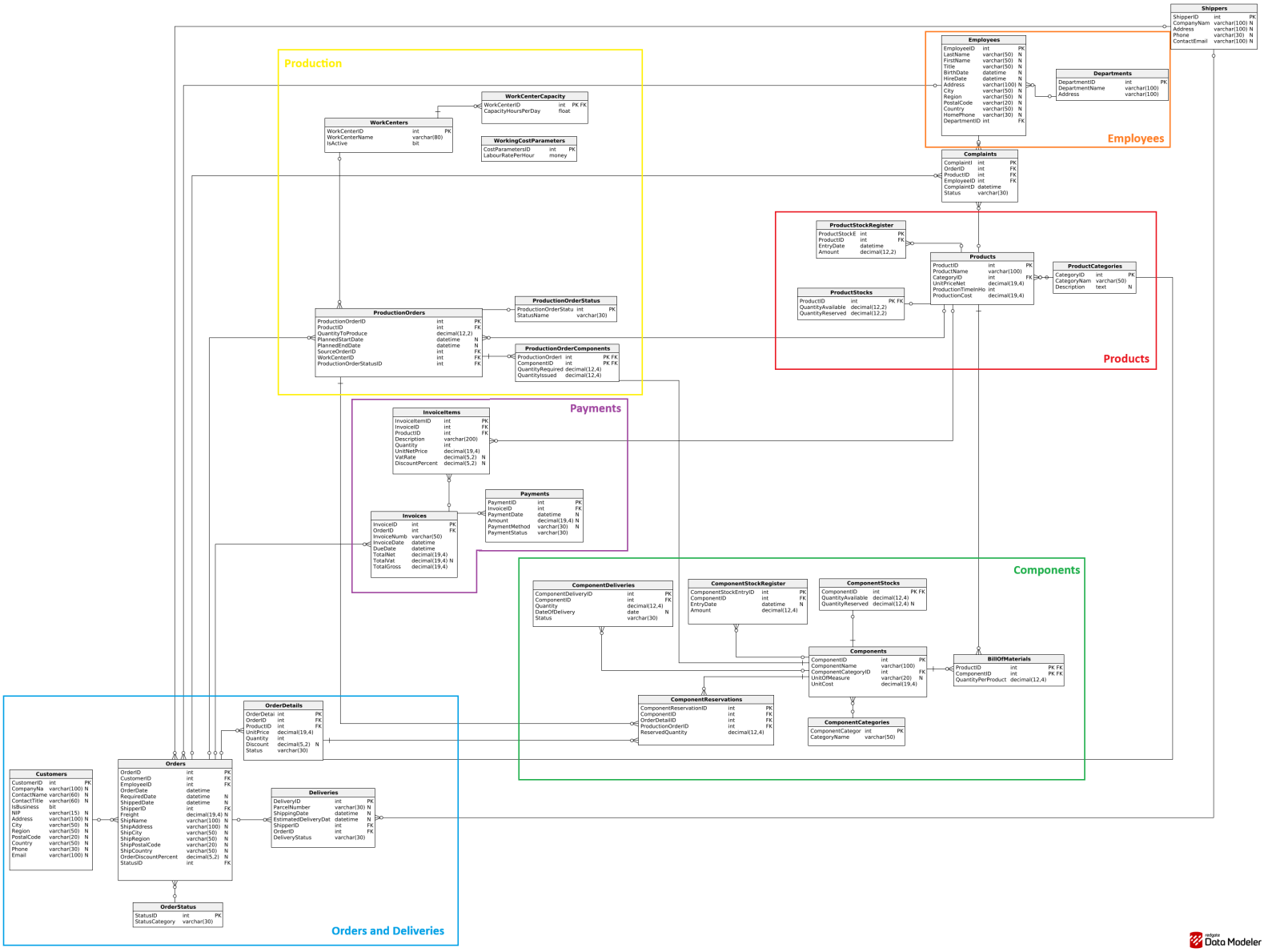
Dokumentacja opisuje szczegółowo strukturę bazy danych, definicje tabel, relacje pomiędzy nimi, warunki integralnościowe oraz kod SQL tworzący wszystkie obiekty systemu, stanowiąc kompletny opis zaprojektowanego rozwiązania.

2. Sprawozdanie

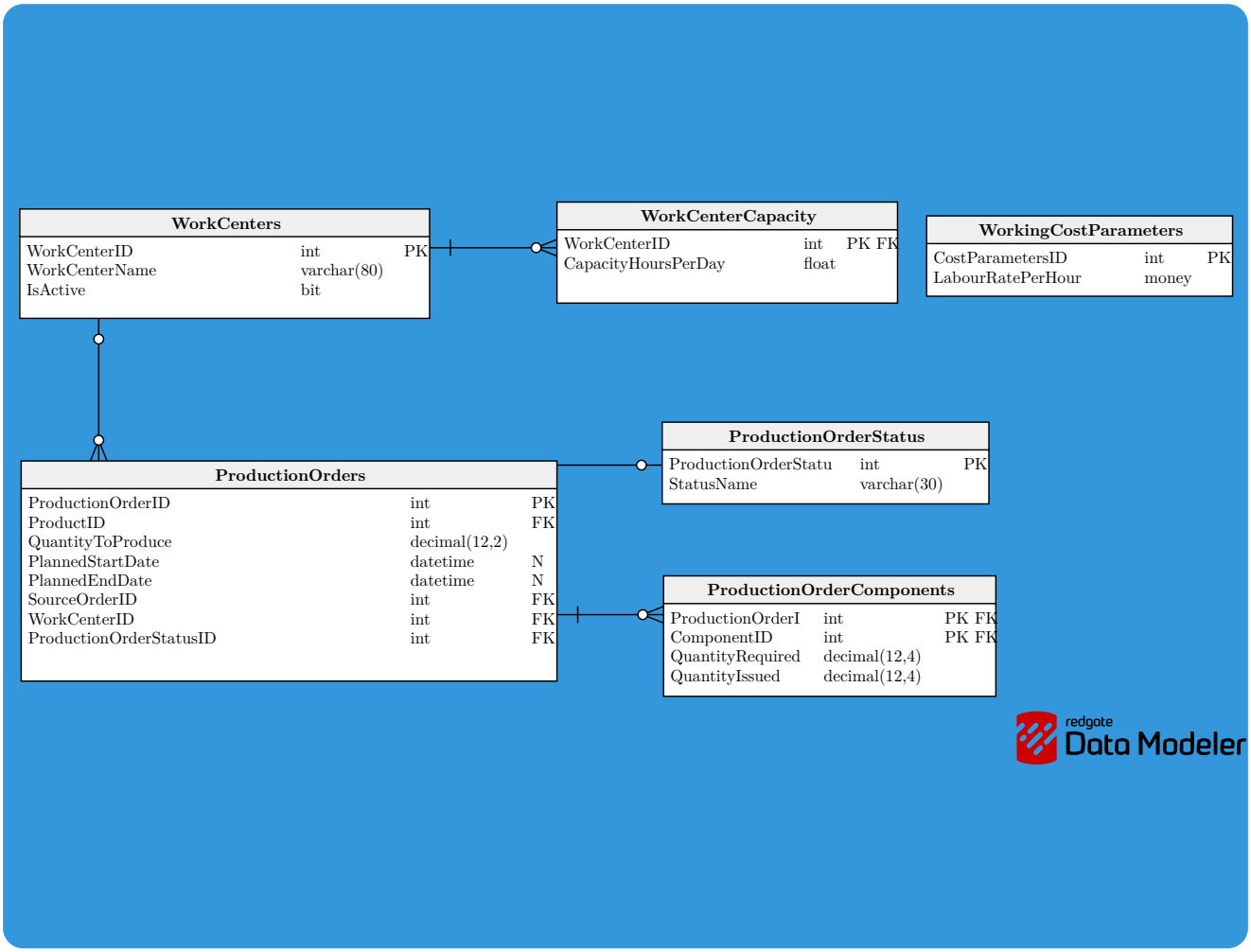
Schemat bazy danych



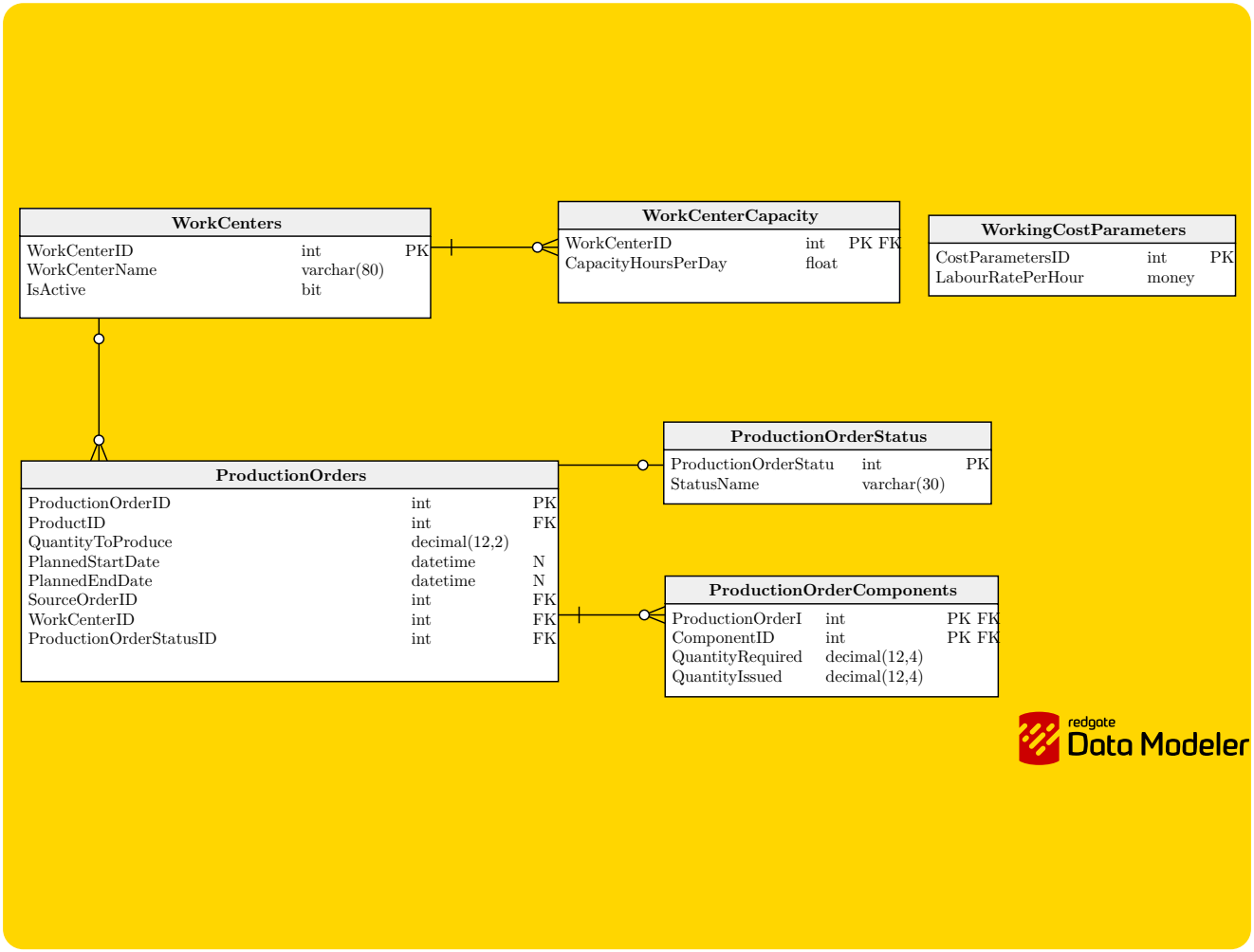
Rysunek 1: Pełny schemat bazy danych



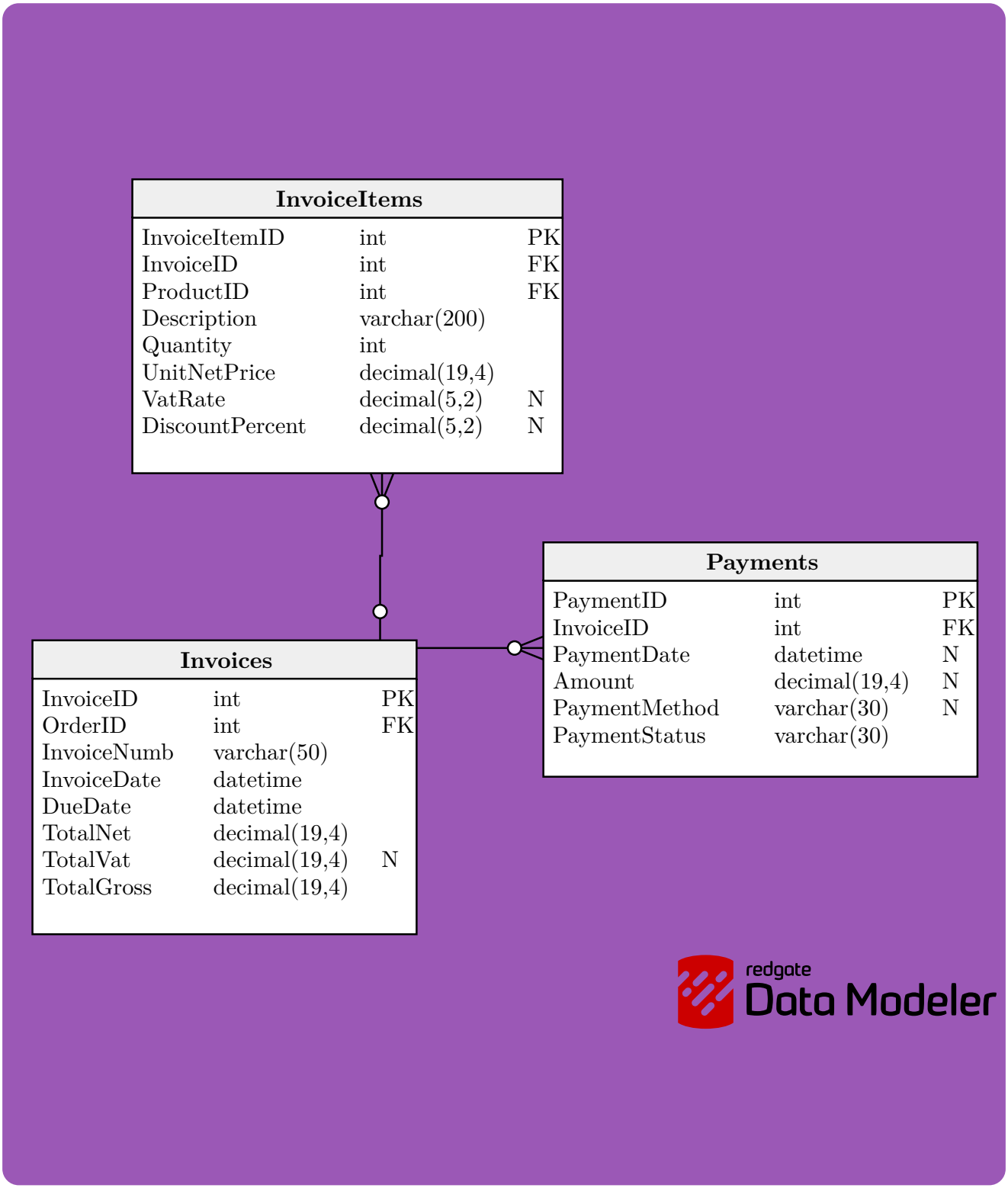
Rysunek 2: Pełny schemat bazy danych wraz z podziałem na sekcje



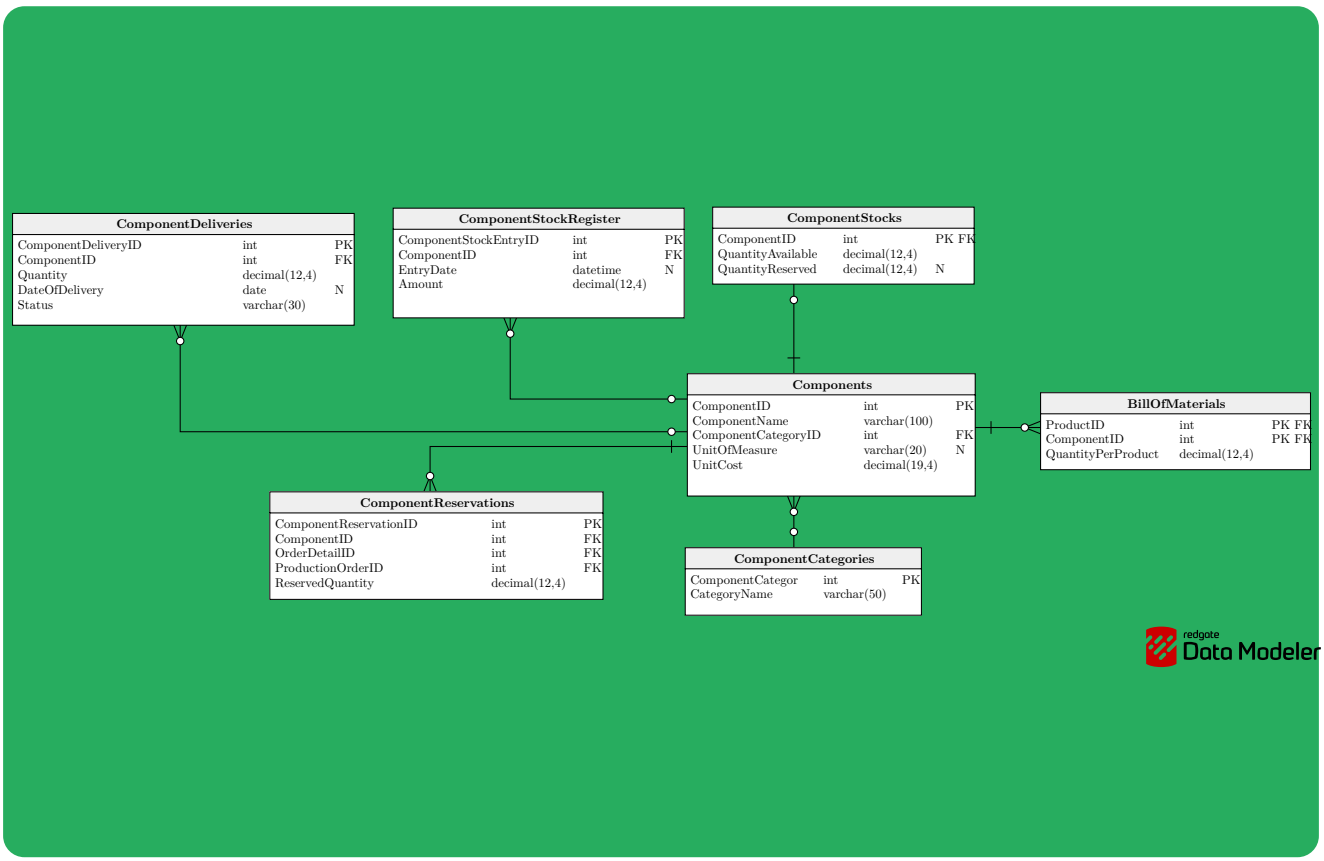
Rysunek 3: Sekcja Orders



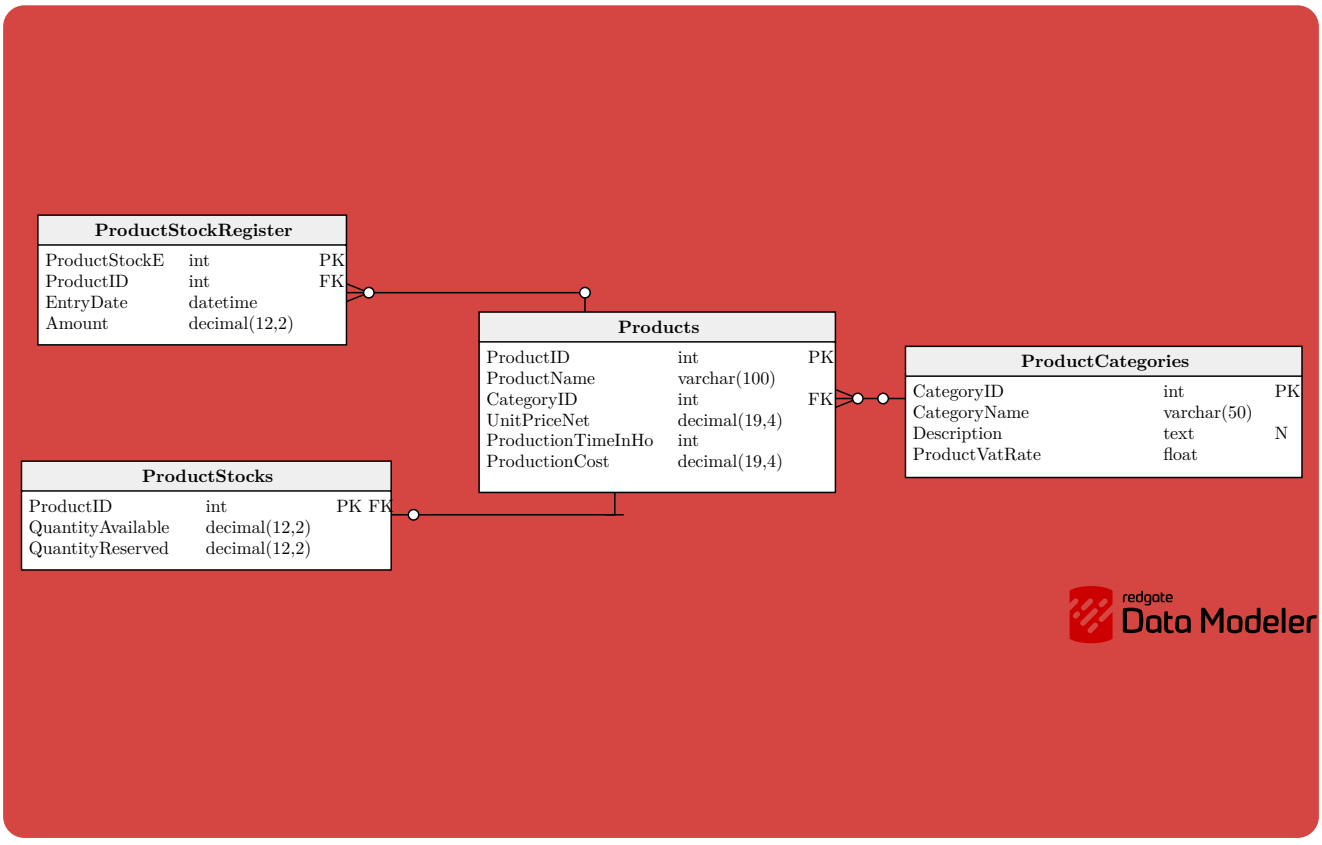
Rysunek 4: Sekcja Production



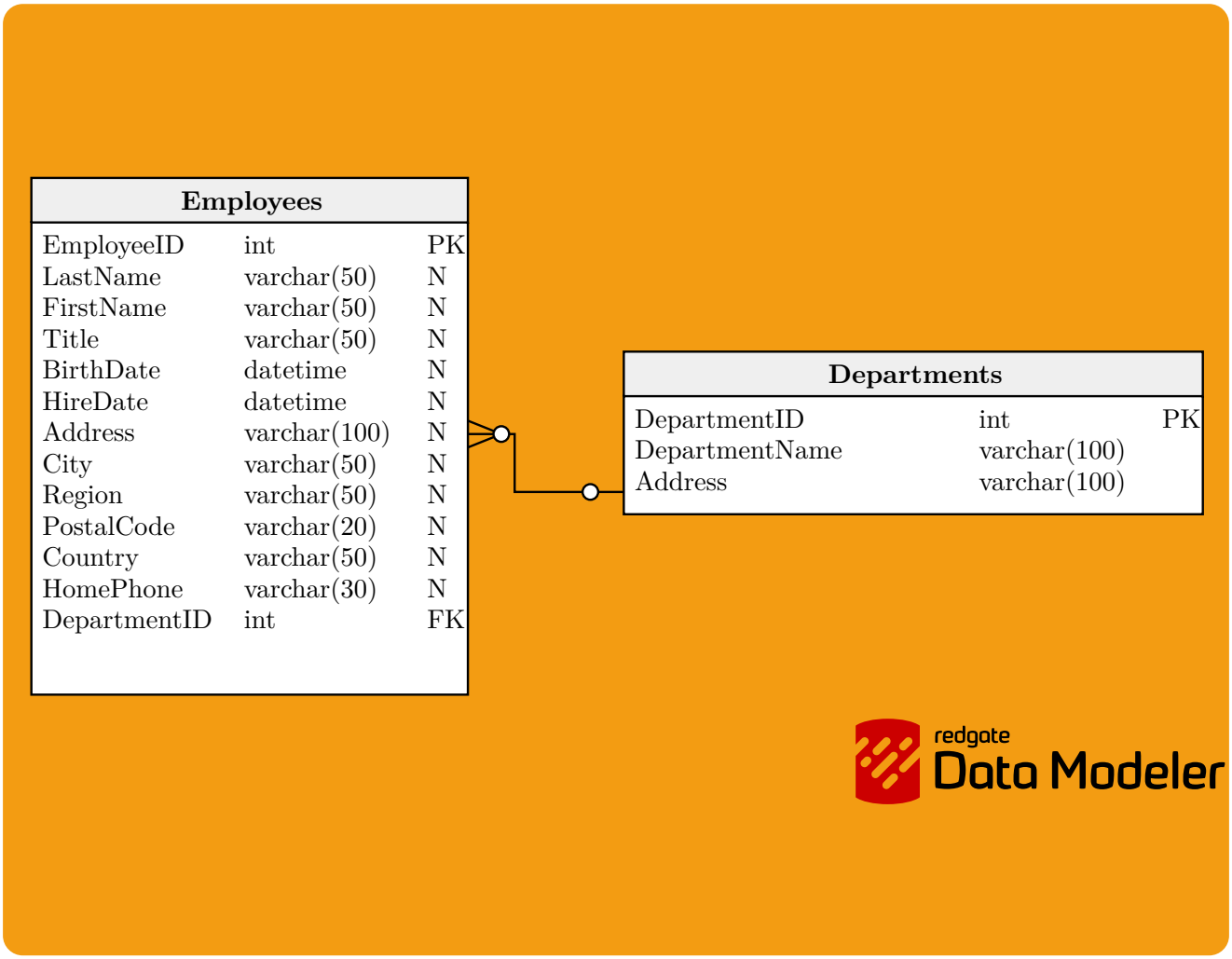
Rysunek 5: Sekcja Payments



Rysunek 6: Sekcja Components



Rysunek 7: Sekcja Products



Rysunek 8: Sekcja Employees

Tabele

Poniżej przedstawiono tabele utworzono w bazie danych. Na tej podstawie wygenerowany został schemat bazy. Służą one do utworzenia bazy. Poniższy kod został przygotowany przy użyciu narzędzia Redgate Data Modeler

BillOfMaterials

Tabela przechowująca informacje o komponentach potrzebnych do wyprodukowania danego produktu oraz ich ilości.

- **ProductID** (int) – identyfikator produktu (klucz obcy → Products.ProductID)
- **ComponentID** (int) – identyfikator komponentu (klucz obcy → Components.ComponentID)
- **QuantityPerProduct** (decimal) – ilość komponentu potrzebna do wyprodukowania jednej sztuki produktu

Table: BillOfMaterials

```
CREATE TABLE BillOfMaterials (
    ProductID int  NOT NULL,
    ComponentID int  NOT NULL,
    QuantityPerProduct decimal(12,4)  NOT NULL,
    CONSTRAINT QuantityPerProduct CHECK (QuantityPerProduct > 0),
    CONSTRAINT BillOfMaterials_pk PRIMARY KEY  (ProductID,ComponentID)
);
```

Complaints

Tabela przechowująca zgłoszenia reklamacyjne dotyczące zamówień i produktów.

- **ComplaintID** (int) – identyfikator reklamacji
- **OrderID** (int) – identyfikator zamówienia, którego dotyczy reklamacja
- **ProductID** (int) – identyfikator reklamowanego produktu
- **EmployeeID** (int) – pracownik obsługujący reklamację
- **ComplaintDate** (datetime) – data zgłoszenia reklamacji

Table: Complaints

```
CREATE TABLE Complaints (
    ComplaintID int  NOT NULL IDENTITY(1, 1),
    OrderID int  NOT NULL,
    ProductID int  NOT NULL,
    EmployeeID int  NOT NULL,
    ComplaintDate datetime  NOT NULL,
    Status varchar(30)  NOT NULL,
    CONSTRAINT Complaints_pk PRIMARY KEY  (ComplaintID)
);
```

ComponentCategories

Tabela przechowująca kategorie komponentów wykorzystywanych w procesie produkcji.

- **ComponentCategoryID** (int) – identyfikator kategorii komponentu
- **CategoryName** (varchar) – nazwa kategorii

Table: ComponentCategories

```
CREATE TABLE ComponentCategories (
    ComponentCategoryID int  NOT NULL IDENTITY(1, 1),
    CategoryName varchar(50)  NOT NULL,
    CONSTRAINT ComponentCategories_pk PRIMARY KEY  (ComponentCategoryID)
);
```

ComponentDeliveries

Tabela rejestrująca dostawy komponentów do magazynu.

- **ComponentDeliveryID** (int) – identyfikator dostawy komponentu
- **ComponentID** (int) – identyfikator dostarczonego komponentu
- **Quantity** (decimal) – ilość dostarczonych komponentów
- **DateOfDelivery** (datetime) – data dostawy

Table: ComponentDeliveries

```
CREATE TABLE ComponentDeliveries (
    ComponentDeliveryID int  NOT NULL IDENTITY(1, 1),
    ComponentID int  NOT NULL,
    Quantity decimal(12,4)  NOT NULL,
    DateOfDelivery date  NULL,
    Status varchar(30)  NOT NULL,
    CONSTRAINT Quantity CHECK (Quantity > 0),
    CONSTRAINT ComponentDeliveries_pk PRIMARY KEY  (ComponentDeliveryID)
);
```

ComponentReservations

Tabela przechowująca informacje o rezerwacjach komponentów na potrzeby zamówień lub produkcji.

- **ComponentReservationID** (int) – identyfikator rezerwacji
- **ComponentID** (int) – identyfikator komponentu
- **OrderDetailID** (int) – pozycja zamówienia powiązana z rezerwacją (opcjonalnie)
- **ProductionOrderID** (int) – zlecenie produkcyjne powiązane z rezerwacją (opcjonalnie)
- **ReservedQuantity** (decimal) – ilość zarezerwowanych komponentów

Table: ComponentReservations

```
CREATE TABLE ComponentReservations (
    ComponentReservationID int  NOT NULL,
    ComponentID int  NOT NULL,
    OrderDetailID int  NOT NULL,
    ProductionOrderID int  NOT NULL,
    ReservedQuantity decimal(12,4)  NOT NULL,
    CONSTRAINT ReservedQuantity CHECK (ReservedQuantity > 0),
    CONSTRAINT ComponentReservations_pk PRIMARY KEY  (ComponentReservationID)
);
```

ComponentStockRegister

Tabela przechowująca historię wszystkich zmian stanów magazynowych komponentów.

- **ComponentStockEntryID** (int) – identyfikator wpisu magazynowego
- **ComponentID** (int) – identyfikator komponentu
- **EntryDate** (datetime) – data operacji magazynowej
- **Amount** (decimal) – zmiana ilości komponentów (wartość dodatnia lub ujemna)

Table: ComponentStockRegister

```
CREATE TABLE ComponentStockRegister (
    ComponentStockEntryID int  NOT NULL IDENTITY(1, 1),
    ComponentID int  NOT NULL,
    EntryDate datetime  NULL,
    Amount decimal(12,4)  NOT NULL,
    CONSTRAINT ComponentStockRegister_pk PRIMARY KEY  (ComponentStockEntryID)
);
```

ComponentStocks

Tabela przechowująca aktualne stany magazynowe komponentów.

- **ComponentID** (int) – identyfikator komponentu
- **QuantityAvailable** (decimal) – ilość dostępna w magazynie
- **QuantityReserved** (decimal) – ilość zarezerwowana

Table: ComponentStocks

```
CREATE TABLE ComponentStocks (
    ComponentID int  NOT NULL,
    QuantityAvailable decimal(12,4)  NOT NULL DEFAULT (0),
    QuantityReserved decimal(12,4)  NULL DEFAULT (0),
    CONSTRAINT QuantityReserved CHECK (QuantityReserved > 0 ),
    CONSTRAINT ComponentStocks_pk PRIMARY KEY  (ComponentID)
);
```

Components

Tabela przechowująca informacje o komponentach wykorzystywanych do produkcji produktów.

- **ComponentID** (int) – identyfikator komponentu
- **ComponentName** (varchar) – nazwa komponentu
- **ComponentCategoryID** (int) – kategoria komponentu
- **UnitOfMeasure** (varchar) – jednostka miary
- **UnitCost** (decimal) – koszt jednostkowy komponentu

Table: Components

```
CREATE TABLE Components (
    ComponentID int  NOT NULL IDENTITY(1, 1),
    ComponentName varchar(100)  NOT NULL,
    ComponentCategoryID int  NOT NULL,
    UnitOfMeasure varchar(20)  NULL,
    UnitCost decimal(19,4)  NOT NULL,
    CONSTRAINT UnitCost CHECK (UnitCost > 0),
    CONSTRAINT Components_pk PRIMARY KEY  (ComponentID)
);
```

Customers

Tabela przechowująca dane klientów indywidualnych oraz firmowych składających zamówienia.

- **CustomerID** (int) – identyfikator klienta
- **CompanyName** (varchar) – nazwa firmy (opcjonalnie)
- **ContactName** (varchar) – osoba kontaktowa
- **ContactTitle** (varchar) – stanowisko osoby kontaktowej
- **IsBusiness** (bit) – informacja, czy klient jest firmą
- **NIP** (varchar) – numer NIP klienta biznesowego
- **Address** (varchar) – adres
- **City** (varchar) – miasto
- **Region** (varchar) – region
- **PostalCode** (varchar) – kod pocztowy
- **Country** (varchar) – kraj
- **Phone** (varchar) – numer telefonu
- **Email** (varchar) – adres e-mail

Table: Customers

```
CREATE TABLE Customers (
    CustomerID int  NOT NULL IDENTITY(1, 1),
    CompanyName varchar(100)  NULL,
    ContactName varchar(60)  NULL,
    ContactTitle varchar(60)  NULL,
    IsBusiness bit  NOT NULL,
    NIP varchar(15)  NULL,
    Address varchar(100)  NULL,
    City varchar(50)  NULL,
    Region varchar(50)  NULL,
    PostalCode varchar(20)  NULL,
    Country varchar(50)  NULL,
    Phone varchar(30)  NULL,
    Email varchar(100)  NULL,
    CONSTRAINT NIP UNIQUE (NIP),
    CONSTRAINT Customers_pk PRIMARY KEY  (CustomerID)
);
```

Deliveries

Tabela przechowująca informacje o dostawach realizowanych dla zamówień klientów.

- **DeliveryID** (int) – identyfikator dostawy
- **ParcelNumber** (varchar) – numer przesyłki
- **ShippingDate** (datetime) – data wysyłki
- **EstimatedDeliveryDate** (datetime) – planowana data dostarczenia
- **ShipperID** (int) – firma realizująca dostawę
- **OrderID** (int) – identyfikator zamówienia
- **DeliveryStatus** (varchar) – status dostawy

Table: Deliveries

```
CREATE TABLE Deliveries (
    DeliveryID int  NOT NULL IDENTITY(1, 1),
    ParcelNumber varchar(30)  NULL,
    ShippingDate datetime  NULL,
    EstimatedDeliveryDate datetime  NULL,
    ShipperID int  NOT NULL,
    OrderID int  NOT NULL,
    DeliveryStatus varchar(30)  NOT NULL,
    CONSTRAINT AK_0 UNIQUE (ParcelNumber),
    CONSTRAINT Deliveries_pk PRIMARY KEY  (DeliveryID)
);
```

Departments

Tabela przechowująca listę działów firmy (np. produkcja, logistyka, sprzedaż).

- **DepartmentID** (int) – identyfikator działu
- **DepartmentName** (varchar(100)) – nazwa działu
- **Address** (varchar(100)) – adres/lokalizacja działu

Table: Departments

```
CREATE TABLE Departments (
    DepartmentID int  NOT NULL IDENTITY(1, 1),
    DepartmentName varchar(100)  NOT NULL,
    Address varchar(100)  NOT NULL,
    CONSTRAINT Departments_pk PRIMARY KEY  (DepartmentID)
);
```

Employees

Tabela przechowująca dane pracowników oraz ich przypisanie do działu.

- **EmployeeID** (int) – identyfikator pracownika
- **LastName** (varchar(50)) – nazwisko pracownika
- **FirstName** (varchar(50)) – imię pracownika
- **Title** (varchar(50)) – stanowisko pracownika
- **BirthDate** (datetime) – data urodzenia pracownika
- **HireDate** (datetime) – data zatrudnienia pracownika
- **Address** (varchar(100)) – adres zamieszkania pracownika
- **City** (varchar(50)) – miasto zamieszkania
- **Region** (varchar(50)) – region/województwo
- **PostalCode** (varchar(20)) – kod pocztowy
- **Country** (varchar(50)) – kraj
- **HomePhone** (varchar(30)) – telefon kontaktowy pracownika
- **DepartmentID** identyfikator działu, do którego przypisany jest pracownik (powiązanie z `Departments`)

Table: Employees

```
CREATE TABLE Employees (
    EmployeeID int  NOT NULL IDENTITY(1, 1),
    LastName varchar(50)  NULL,
    FirstName varchar(50)  NULL,
    Title varchar(50)  NULL,
    BirthDate datetime  NULL,
    HireDate datetime  NULL,
    Address varchar(100)  NULL,
    City varchar(50)  NULL,
    Region varchar(50)  NULL,
    PostalCode varchar(20)  NULL,
    Country varchar(50)  NULL,
    HomePhone varchar(30)  NULL,
    DepartmentID int  NOT NULL,
    CONSTRAINT Employees_pk PRIMARY KEY  (EmployeeID)
);
```

InvoiceItems

Tabela przechowująca szczegółowe pozycje znajdujące się na fakturach sprzedaży.

- **InvoiceItemID (int)** – unikalny identyfikator pozycji faktury
- **InvoiceID (int)** – identyfikator faktury, do której należy dana pozycja (powiązanie z tabelą `Invoices`)
- **ProductID (int)** – identyfikator produktu objętego pozycją faktury (powiązanie z tabelą `Products`)
- **Description (varchar)** – opis pozycji na fakturze (np. nazwa handlowa produktu lub dodatkowe informacje)
- **Quantity (decimal)** – liczba sprzedanych sztuk produktu
- **UnitNetPrice (decimal)** – cena jednostkowa netto produktu
- **VatRate (float)** – stawka podatku VAT obowiązująca dla danej pozycji
- **DiscountPercent (float)** – procentowy rabat zastosowany do pozycji

Table: InvoiceItems

```
CREATE TABLE InvoiceItems (
    InvoiceItemID int NOT NULL IDENTITY(1,1),
    InvoiceID int NOT NULL,
    ProductID int NOT NULL,
    Description varchar(200) NOT NULL,
    Quantity int NOT NULL,
    UnitNetPrice decimal(19,4) NOT NULL,
    VatRate decimal(5,2) NULL,
    DiscountPercent decimal(5,2) NULL DEFAULT (0),

    CONSTRAINT CK_InvoiceItems_Quantity
        CHECK (Quantity > 0),

    CONSTRAINT CK_InvoiceItems_UnitNetPrice
        CHECK (UnitNetPrice > 0),

    CONSTRAINT PK_InvoiceItems
        PRIMARY KEY (InvoiceItemID)
);
```

Invoices

Tabela przechowująca podstawowe dane faktur sprzedaży wystawianych do zamówień klientów.

- **InvoiceID (int)** – identyfikator faktury (klucz główny)
- **OrderID (int)** – identyfikator zamówienia, którego dotyczy faktura (powiązanie z tabelą `Orders`)
- **InvoiceNumber (varchar)** – numer faktury
- **InvoiceDate (datetime)** – data wystawienia faktury
- **DueDate (datetime)** – termin płatności faktury
- **TotalNet (decimal)** – łączna wartość netto faktury
- **TotalVat (decimal)** – łączna kwota podatku VAT
- **TotalGross (decimal)** – łączna wartość brutto faktury

Table: Invoices

```
CREATE TABLE Invoices (
    InvoiceID int  NOT NULL IDENTITY(1, 1),
    OrderID int  NOT NULL,
    InvoiceNumber varchar(50)  NOT NULL,
    InvoiceDate datetime  NOT NULL,
    DueDate datetime  NOT NULL,
    TotalNet decimal(19,4)  NOT NULL,
    TotalVat decimal(19,4)  NULL,
    TotalGross decimal(19,4)  NOT NULL,
    CONSTRAINT InvoiceNumber UNIQUE (InvoiceNumber),
    CONSTRAINT TotalNet CHECK (TotalNet > 0),
    CONSTRAINT TotalGross CHECK (TotalGross > 0),
    CONSTRAINT Invoices_pk PRIMARY KEY  (InvoiceID)
);
```

OrderDetails

Tabela przechowująca szczegółowe pozycje zamówień klientów.

- **OrderDetailID (int)** – identyfikator pozycji zamówienia
- **OrderID (int)** – identyfikator zamówienia (powiązanie z tabelą `Orders`)
- **ProductID (int)** – identyfikator zamówionego produktu (powiązanie z tabelą `Products`)
- **UnitPrice (decimal)** – cena jednostkowa produktu w momencie składania zamówienia
- **Quantity (decimal)** – liczba zamówionych sztuk produktu
- **Discount (float)** – rabat udzielony dla danej pozycji zamówienia

Table: OrderDetails

```
CREATE TABLE OrderDetails (
    OrderDetailID int  NOT NULL IDENTITY(1, 1),
    OrderID int  NOT NULL,
    ProductID int  NOT NULL,
    UnitPrice decimal(19,4)  NOT NULL,
    Quantity int  NOT NULL,
    Discount decimal(5,2)  NULL DEFAULT (0),
    Status varchar(30)  NOT NULL,
    CONSTRAINT CK_OrderDetails_Quantity
        CHECK (Quantity > 0),

    CONSTRAINT CK_OrderDetails_UnitPrice
        CHECK (UnitPrice > 0),

    CONSTRAINT PK_OrderDetails
        PRIMARY KEY (OrderDetailID)
);
```

OrderStatus

Tabela przechowująca możliwe stany realizacji zamówień klientów.

- **StatusID (int)** – identyfikator statusu zamówienia
- **StatusCategory (varchar)** – nazwa lub kategoria statusu (np. „Nowe”, „W realizacji”, „Wysłane”, „Zakończone”, „Anulowane”)

Table: OrderStatus

```
CREATE TABLE OrderStatus (
    StatusID int  NOT NULL IDENTITY(1, 1),
    StatusCategory varchar(30)  NOT NULL,
    CONSTRAINT OrderStatus_pk PRIMARY KEY  (StatusID)
);
```

Orders

Tabela przechowująca zamówienia składane przez klientów.

- **OrderID (int)** – identyfikator zamówienia
- **CustomerID (int)** – identyfikator klienta składającego zamówienie
- **EmployeeID (int)** – identyfikator pracownika obsługującego zamówienie
- **OrderDate (datetime)** – data złożenia zamówienia
- **RequiredDate (datetime)** – oczekiwana data realizacji zamówienia
- **ShippedDate (datetime)** – data wysyłki zamówienia
- **ShipperID (int)** – identyfikator firmy przewozowej realizującej dostawę
- **Freight (decimal)** – koszt transportu zamówienia
- **ShipName (varchar)** – nazwa odbiorcy przesyłki
- **ShipAddress (varchar)** – adres dostawy
- **ShipCity (varchar)** – miasto dostawy
- **ShipRegion (varchar)** – region/województwo dostawy
- **ShipPostalCode (varchar)** – kod pocztowy dostawy
- **ShipCountry (varchar)** – kraj dostawy
- **OrderDiscountPercent (float)** – rabat procentowy zastosowany do całego zamówienia
- **StatusID (int)** – identyfikator aktualnego statusu zamówienia (powiązanie z tabelą `OrderStatus`)

Table: Orders

```
CREATE TABLE Orders (
    OrderID int  NOT NULL IDENTITY(1, 1),
    CustomerID int  NOT NULL,
    EmployeeID int  NOT NULL,
    OrderDate datetime  NOT NULL,
    RequiredDate datetime  NULL,
    ShippedDate datetime  NULL,
    ShipperID int  NOT NULL,
    Freight decimal(19,4)  NULL,
    ShipName varchar(100)  NULL,
    ShipAddress varchar(100)  NULL,
    ShipCity varchar(50)  NULL,
    ShipRegion varchar(50)  NULL,
    ShipPostalCode varchar(20)  NULL,
    ShipCountry varchar(50)  NULL,
    OrderDiscountPercent decimal(5,2)  NULL DEFAULT (0),
    StatusID int  NOT NULL,
    CONSTRAINT Orders_pk PRIMARY KEY  (OrderID)
);
```

Payments

Tabela przechowująca informacje o płatnościach dokonanych za faktury.

- **PaymentID (int)** – identyfikator płatności
- **InvoiceID (int)** – identyfikator faktury, której dotyczy płatność (powiązanie z tabelą `Invoices`)
- **PaymentDate (datetime)** – data dokonania płatności
- **Amount (decimal)** – kwota zapłaty
- **PaymentMethod (varchar)** – metoda płatności (np. przelew, karta, gotówka)
- **PaymentStatus (varchar)** – status płatności (np. „oczekująca”, „zaksięgowana”, „odrzucona”)

Table: Payments

```
CREATE TABLE Payments (
    PaymentID int  NOT NULL IDENTITY(1, 1),
    InvoiceID int  NOT NULL,
    PaymentDate datetime  NULL,
    Amount decimal(19,4)  NULL,
    PaymentMethod varchar(30)  NULL,
    PaymentStatus varchar(30)  NOT NULL,
    CONSTRAINT Payments_pk PRIMARY KEY  (PaymentID)
);
```

ProductCategories

Tabela kategorii produktów oferowanych przez firmę.

- **CategoryID (int)** – identyfikator kategorii produktu
- **CategoryName (varchar)** – nazwa kategorii (np. „Stoły", „Krzesła", „Szafy")
- **Description (text)** – opis kategorii produktów
- **ProductVatRate (float)** – domyślna stawka VAT przypisana do produktów z danej kategorii

Table: ProductCategories

```
CREATE TABLE ProductCategories (
    CategoryID int  NOT NULL IDENTITY(1, 1),
    CategoryName varchar(50)  NOT NULL,
    Description text  NULL,
    ProductVatRate float  NOT NULL,
    CONSTRAINT ProductCategories_pk PRIMARY KEY  (CategoryID)
);
```

ProductStockRegister

Tabela rejestrująca historię zmian stanów magazynowych produktów.

- **ProductStockEntryID (int)** – identyfikator wpisu w rejestrze magazynowym
- **ProductID (int)** – identyfikator produktu, którego dotyczy wpis (powiązanie z tabelą `Products`)
- **EntryDate (datetime)** – data i czas wykonania operacji magazynowej
- **Amount (decimal)** – zmiana ilości produktu (wartość dodatnia – przyjęcie, ujemna – wydanie)

Table: ProductStockRegister

```
CREATE TABLE ProductStockRegister (
    ProductStockEntryID int  NOT NULL IDENTITY(1, 1),
    ProductID int  NOT NULL,
    EntryDate datetime  NOT NULL,
    Amount decimal(12,2)  NOT NULL,
    CONSTRAINT ProductStockRegister_pk PRIMARY KEY  (ProductStockEntryID)
);
```

ProductStocks

Tabela pokazująca stan magazynu produktów.

- **ProductID** (int) - identyfikator produktu
- **QuantityAvailable** (decimal(12,2)) - liczba dostępnych sztuk produktu
- **QuantityReserved** (decimal(12,2)) - liczba zarezerwowanych sztuk produktu

Table: ProductStocks

```
CREATE TABLE ProductStocks (
    ProductID int  NOT NULL,
    QuantityAvailable decimal(12,2)  NOT NULL DEFAULT (0),
    QuantityReserved decimal(12,2)  NOT NULL DEFAULT (0),
    CONSTRAINT CK_ProductStocks_QuantityReserved
        CHECK (QuantityReserved >= 0),

    CONSTRAINT PK_ProductStocks
        PRIMARY KEY (ProductID)
);
```

ProductionOrderComponents

Tabela przedstawiająca dokładną zawartość zamówienia na produkcję.

- **ProductionOrderID** (int) - identyfikator zamówienia produkcyjnego
- **ComponentID** (int) - identyfikator komponentu
- **QuantityRequired** (decimal(12,4)) - wymagana liczba sztuk materiału
- **QuantityIssued** (decimal(12,4)) - liczba wydanych sztuk materiału

Table: ProductionOrderComponents

```
CREATE TABLE ProductionOrderComponents (
    ProductionOrderID int  NOT NULL,
    ComponentID int  NOT NULL,
    QuantityRequired decimal(12,4)  NOT NULL,
    QuantityIssued decimal(12,4)  NOT NULL DEFAULT (0),
    CONSTRAINT QuantityRequired CHECK (QuantityRequired > 0),
    CONSTRAINT ProductionOrderComponents_pk PRIMARY KEY
(ProductionOrderID,ComponentID)
);
```

ProductionOrderStatus

Tabela określająca stany zamówień produkcyjnych.

- **ProductionOrderStatusID** (int) - identyfikator stanu zamówienia produkcyjnego
- **StatusName** (varchar(30)) - nazwa stanu

Table: ProductionOrderStatus

```
CREATE TABLE ProductionOrderStatus (
    ProductionOrderStatusID int  NOT NULL,
    StatusName varchar(30)  NOT NULL,
    CONSTRAINT ProductionOrderStatusID_pk PRIMARY KEY  (ProductionOrderStatusID)
);
```

ProductionOrders

Tabela zestawiająca obecne zamówienia produkcyjne.

- **ProductionOrderID** (int) - identyfikator zamówienia produkcyjnego
- **ProductID** (int) - identyfikator produktu
- **QuantityToProduce** (decimal(12,2)) - liczba produktów do wyprodukowania
- **PlannedStartDate** (datetime) - planowany czas rozpoczęcia realizacji zamównienia
- **PlannedEndDate** (datetime) - planowany czas zakończenia realizacji zamównienia
- **SourceOrderID** (int) - identyfikator zamówienia klienta, którego pokrycie ma zapewnić to zamówienie produkcyjne
- **WorkCenterID** (int) - identyfikator zakładu, w którym zostanie zrealizowane zamówienie
- **ProductionOrderStatusID** (int) - identyfikator stanu zamówienia produkcyjnego

Table: ProductionOrders

```
CREATE TABLE ProductionOrders (
    ProductionOrderID int  NOT NULL IDENTITY(1, 1),
    ProductID int  NOT NULL,
    QuantityToProduce decimal(12,2)  NOT NULL,
    PlannedStartDate datetime  NULL,
    PlannedEndDate datetime  NULL,
    SourceOrderID int  NOT NULL,
    WorkCenterID int  NOT NULL,
    ProductionOrderStatusID int  NOT NULL,
    CONSTRAINT QuantityToProduce CHECK (QuantityToProduce > 0 ),
    CONSTRAINT ProductionOrders_pk PRIMARY KEY  (ProductionOrderID)
);
```

Products

Tabela przedstawiająca oferowane produkty.

- **ProductID** (int) - identyfikator produktu
- **ProductName** (varchar(100)) - nazwa produktu
- **CategoryID** (int) - identyfikator kategorii produktu
- **UnitPriceNet** (decimal(19,4)) - cena netto za dany produkt
- **ProductionTimeInHours** (int) - czas produkcji produktu w godzinach
- **ProductionCost** (decimal(19,4)) - koszt produkcji produktu

Table: Products

```
CREATE TABLE Products (
    ProductID int  NOT NULL IDENTITY(1, 1),
    ProductName varchar(100)  NOT NULL,
    CategoryID int  NOT NULL,
    UnitPriceNet decimal(19,4)  NOT NULL,
    ProductionTimeInHours int  NOT NULL,
    ProductionCost decimal(19,4)  NOT NULL,
    CONSTRAINT UnitPriceNet CHECK (UnitPriceNet > 0),
    CONSTRAINT ProductionTime CHECK (ProductionTimeInHours > 0),
    CONSTRAINT ProductionCost CHECK (ProductionCost > 0),
    CONSTRAINT Products_pk PRIMARY KEY  (ProductID)
);
```

Shippers

Tabela zestawiająca przewoźników.

- **ShipperID** (int) - identyfikator przewoźnika
- **CompanyName** (varchar(100)) - nazwa firmy spedycyjnej
- **Address** (varchar(100)) - adres firmy
- **Phone** (varchar(30)) - telefon do firmy
- **ContactEmail** (varchar(100)) - email kontaktowy firmy

Table: Shippers

```
CREATE TABLE Shippers (
    ShipperID int  NOT NULL IDENTITY(1, 1),
    CompanyName varchar(100)  NULL,
    Address varchar(100)  NULL,
    Phone varchar(30)  NULL,
    ContactEmail varchar(100)  NULL,
    CONSTRAINT Shippers_pk PRIMARY KEY  (ShipperID)
);
```

WorkCenterCapacity

Tabela opisująca zdolność produkcji danego zakładu produkcyjnego.

- **WorkCenterID** (int) - identyfikator zakładu
- **CapacityHoursPerDay** (float) - liczba godzin w ciągu dnia, kiedy zakład jest czynny (i wytwarza materiały)

Table: WorkCenterCapacity

```
CREATE TABLE WorkCenterCapacity (
    WorkCenterID int  NOT NULL,
    CapacityHoursPerDay float  NOT NULL,
    CONSTRAINT work_center_id PRIMARY KEY  (WorkCenterID)
);
```

WorkCenters

Tabela zestawiająca zakłady produkcyjne.

- **WorkCenterID** (int) - identyfikator zakładu
- **WorkCenterName** (varchar(80)) - nazwa zakładu
- **IsActive** (bit) - flaga logiczna określająca, czy zakład jest czynny

Table: WorkCenters

```
CREATE TABLE WorkCenters (
    WorkCenterID int  NOT NULL,
    WorkCenterName varchar(80)  NOT NULL,
    IsActive bit  NOT NULL,
    CONSTRAINT UQ_WorkCenters_Name UNIQUE (WorkCenterName),
    CONSTRAINT WorkCenters_pk PRIMARY KEY  (WorkCenterID)
);
```

WorkingCostParameters

Tabela zestawiająca koszty pracy w zakładach.

- **CostParametersID** (int) - identyfikator parametrów kosztów pracy w zakładach
- **LabourRatePerHour** (money) - stawka godzinowa za pracę w zakładzie

Table: WorkingCostParameters

```
CREATE TABLE WorkingCostParameters (
    CostParametersID int  NOT NULL,
    LabourRatePerHour money  NOT NULL,
    CONSTRAINT cost_parameters_pk PRIMARY KEY  (CostParametersID)
);
```

Klucze obce

ComponentReservations_Components (table: ComponentReservations)

```
ALTER TABLE ComponentReservations ADD CONSTRAINT ComponentReservations_Components
    FOREIGN KEY (ComponentID)
    REFERENCES Components (ComponentID);
```

ComponentReservations_OrderDetails (table: ComponentReservations)

```
ALTER TABLE ComponentReservations ADD CONSTRAINT
ComponentReservations_OrderDetails
    FOREIGN KEY (OrderDetailID)
    REFERENCES OrderDetails (OrderDetailID);
```

ComponentReservations_ProductionOrders (table: ComponentReservations)

```
ALTER TABLE ComponentReservations ADD CONSTRAINT
ComponentReservations_ProductionOrders
    FOREIGN KEY (ProductionOrderID)
    REFERENCES ProductionOrders (ProductionOrderID);
```

FK_0 (table: Employees)

```
ALTER TABLE Employees ADD CONSTRAINT FK_0
    FOREIGN KEY (DepartmentID)
    REFERENCES Departments (DepartmentID);
```

FK_1 (table: Products)

```
ALTER TABLE Products ADD CONSTRAINT FK_1
    FOREIGN KEY (CategoryID)
    REFERENCES ProductCategories (CategoryID);
```

FK_10 (table: Orders)

```sql
ALTER TABLE Orders ADD CONSTRAINT FK_10
    FOREIGN KEY (CustomerID)
    REFERENCES Customers (CustomerID);
```

FK_11 (table: Orders)

```sql
ALTER TABLE Orders ADD CONSTRAINT FK_11
    FOREIGN KEY (EmployeeID)
    REFERENCES Employees (EmployeeID);
```

FK_12 (table: Orders)

```sql
ALTER TABLE Orders ADD CONSTRAINT FK_12
    FOREIGN KEY (ShipperID)
    REFERENCES Shippers (ShipperID);

## FK_13 (table: Orders)
``` sql
ALTER TABLE Orders ADD CONSTRAINT FK_13
    FOREIGN KEY (StatusID)
    REFERENCES OrderStatus (StatusID);
```

FK_14 (table: OrderDetails)

```sql
ALTER TABLE OrderDetails ADD CONSTRAINT FK_14
    FOREIGN KEY (OrderID)
    REFERENCES Orders (OrderID);
```

FK_15 (table: OrderDetails)

```sql
ALTER TABLE OrderDetails ADD CONSTRAINT FK_15
    FOREIGN KEY (ProductID)
    REFERENCES Products (ProductID);
```

FK_16 (table: Complaints)

```sql
ALTER TABLE Complaints ADD CONSTRAINT FK_16
    FOREIGN KEY (OrderID)
    REFERENCES Orders (OrderID);
```
```

FK_17 (table: Complaints)

```
ALTER TABLE Complaints ADD CONSTRAINT FK_17
    FOREIGN KEY (ProductID)
    REFERENCES Products (ProductID);
```

FK_18 (table: Complaints)

```
ALTER TABLE Complaints ADD CONSTRAINT FK_18
    FOREIGN KEY (EmployeeID)
    REFERENCES Employees (EmployeeID);
```

FK_19 (table: Deliveries)

```
ALTER TABLE Deliveries ADD CONSTRAINT FK_19
    FOREIGN KEY (ShipperID)
    REFERENCES Shippers (ShipperID);
```

FK_2 (table: ProductStocks)

```
ALTER TABLE ProductStocks ADD CONSTRAINT FK_2
    FOREIGN KEY (ProductID)
    REFERENCES Products (ProductID);
```

FK_20 (table: Deliveries)

```
ALTER TABLE Deliveries ADD CONSTRAINT FK_20
    FOREIGN KEY (OrderID)
    REFERENCES Orders (OrderID);
```

FK_21 (table: Invoices)

```
ALTER TABLE Invoices ADD CONSTRAINT FK_21
    FOREIGN KEY (OrderID)
    REFERENCES Orders (OrderID);
```

FK_22 (table: InvoiceItems)

```
ALTER TABLE InvoiceItems ADD CONSTRAINT FK_22
    FOREIGN KEY (InvoiceID)
    REFERENCES Invoices (InvoiceID);
```

FK_23 (table: InvoiceItems)

```
ALTER TABLE InvoiceItems ADD CONSTRAINT FK_23
    FOREIGN KEY (ProductID)
    REFERENCES Products (ProductID);
```

FK_24 (table: Payments)

```
ALTER TABLE Payments ADD CONSTRAINT FK_24
    FOREIGN KEY (InvoiceID)
    REFERENCES Invoices (InvoiceID);
```

FK_25 (table: ProductionOrders)

```
ALTER TABLE ProductionOrders ADD CONSTRAINT FK_25
    FOREIGN KEY (ProductID)
    REFERENCES Products (ProductID);
```

FK_26 (table: ProductionOrders)

```
ALTER TABLE ProductionOrders ADD CONSTRAINT FK_26
    FOREIGN KEY (SourceOrderID)
    REFERENCES Orders (OrderID);
```

FK_27 (table: ProductionOrderComponents)

```
ALTER TABLE ProductionOrderComponents ADD CONSTRAINT FK_27
    FOREIGN KEY (ProductionOrderID)
    REFERENCES ProductionOrders (ProductionOrderID);
```

FK_28 (table: ProductionOrderComponents)

```
ALTER TABLE ProductionOrderComponents ADD CONSTRAINT FK_28
    FOREIGN KEY (ComponentID)
    REFERENCES Components (ComponentID);
```

FK_3 (table: ProductStockRegister)

```
ALTER TABLE ProductStockRegister ADD CONSTRAINT FK_3
    FOREIGN KEY (ProductID)
    REFERENCES Products (ProductID);
```

FK_4 (table: Components)

```
ALTER TABLE Components ADD CONSTRAINT FK_4
    FOREIGN KEY (ComponentCategoryID)
    REFERENCES ComponentCategories (ComponentCategoryID);
```

FK_5 (table: ComponentStocks)

```
ALTER TABLE ComponentStocks ADD CONSTRAINT FK_5
    FOREIGN KEY (ComponentID)
    REFERENCES Components (ComponentID);
```

FK_6 (table: ComponentStockRegister)

```
ALTER TABLE ComponentStockRegister ADD CONSTRAINT FK_6
    FOREIGN KEY (ComponentID)
    REFERENCES Components (ComponentID);
```

FK_7 (table: ComponentDeliveries)

```
ALTER TABLE ComponentDeliveries ADD CONSTRAINT FK_7
    FOREIGN KEY (ComponentID)
    REFERENCES Components (ComponentID);
```

FK_8 (table: BillOfMaterials)

```
ALTER TABLE BillOfMaterials ADD CONSTRAINT FK_8
    FOREIGN KEY (ProductID)
    REFERENCES Products (ProductID);
```

FK_9 (table: BillOfMaterials)

```
ALTER TABLE BillOfMaterials ADD CONSTRAINT FK_9
    FOREIGN KEY (ComponentID)
    REFERENCES Components (ComponentID);
```

ProductionOrders_ProductionOrderStatus (table: ProductionOrders)

```
ALTER TABLE ProductionOrders ADD CONSTRAINT ProductionOrders_ProductionOrderStatus
    FOREIGN KEY (ProductionOrderStatusID)
    REFERENCES ProductionOrderStatus (ProductionOrderStatusID);
```

ProductionOrders_WorkCenters (table: ProductionOrders)

```
ALTER TABLE ProductionOrders ADD CONSTRAINT ProductionOrders_WorkCenters
    FOREIGN KEY (WorkCenterID)
    REFERENCES WorkCenters (WorkCenterID);
```

WorkCenterCapacity_WorkCenters (table: WorkCenterCapacity)

```
ALTER TABLE WorkCenterCapacity ADD CONSTRAINT WorkCenterCapacity_WorkCenters
    FOREIGN KEY (WorkCenterID)
    REFERENCES WorkCenters (WorkCenterID);
```

OPIS SCHEMATU BAZY DANYCH

Schemat bazy danych został zaprojektowany na potrzeby firmy produkcyjno-usługowej zajmującej się produkcją oraz sprzedażą mebli. Model obejmuje pełny cykl działalności przedsiębiorstwa, w tym sprzedaż, planowanie produkcji, gospodarkę magazynową, logistykę oraz rozliczenia finansowe.

---

1. SPRZEDAŻ I OBSŁUGA KLIENTA

Obsługa sprzedaży realizowana jest poprzez następujące tabele:

- **Customers**
- **Orders**
- **OrderDetails**

Umożliwiają one ewidencję klientów indywidualnych i biznesowych, składanie zamówień oraz przechowywanie szczegółowych pozycji zamówień wraz z cenami i rabatami. Statusy zamówień są kontrolowane za pomocą tabeli **OrderStatus**. Proces dostawy zamówień obsługują tabele **Deliveries** oraz **Shippers**.

---

2. FINANSE

Rozliczenia finansowe realizowane są przy użyciu tabel:

- **Invoices**
- **InvoiceItems**
- **Payments**

System umożliwia wystawianie faktur, rejestrowanie ich pozycji oraz ewidencjonowanie płatności dokonywanych przez klientów. Zapewniono jednoznaczną identyfikację dokumentów finansowych poprzez unikalność numerów faktur.

---

3. PRODUKCJA

Część produkcyjna systemu oparta jest na tabelach:

- **Products**
- **ProductCategories**
- **Components**
- **ComponentCategories**
- **BillOfMaterials**

Tabela **BillOfMaterials** określa strukturę materiałową produktu oraz ilości komponentów niezbędnych do jego wytworzenia.

---

4. PLANOWANIE PRODUKCJI I MOCE PRZEROBOWE

Planowanie i realizacja produkcji odbywa się przy użyciu tabel:

- **ProductionOrders**
- **ProductionOrderComponents**
- **ProductionOrderStatus**
- **WorkCenters**
- **WorkCenterCapacity**

Zlecenia produkcyjne mogą być powiązane z zamówieniami klientów oraz przypisane do centrów roboczych, dla których zdefiniowano moce przerobowe w jednostce czasu.

---

5. MAGAZYN

Gospodarka magazynowa została odwzorowana za pomocą tabel:

- **ProductStocks**
- **ProductStockRegister**
- **ComponentStocks**
- **ComponentStockRegister**
- **ComponentDeliveries**
- **ComponentReservations**

System umożliwia kontrolę bieżących stanów magazynowych, rejestrację ruchów magazynowych oraz rezerwację komponentów na potrzeby realizacji zamówień i produkcji.

---

6. STRUKTURA ORGANIZACYJNA I REKLAMACJE

Struktura organizacyjna przedsiębiorstwa została odwzorowana w tabelach:

- **Employees**
- **Departments**

Obsługa reklamacji klientów realizowana jest przy użyciu tabeli **Complaints**.

---

7. INTEGRALNOŚĆ DANYCH

W bazie danych zastosowano następujące mechanizmy integralności:

- klucze główne zapewniające integralność encji,
- klucze obce zapewniające integralność referencyjną,
- ograniczenia **CHECK** kontrolujące poprawność danych,
- ograniczenia **UNIQUE** dla wybranych atrybutów

Zastosowane mechanizmy integralności zapewniają spójność i poprawność danych oraz umożliwiają realizację założeń funkcjonalnych systemu.

WARUNKI INTEGRALNOŚCIOWE W SCHEMACIE BAZY DANYCH

1. Integralność encji (klucze główne)

Każda tabela w schemacie bazy danych posiada klucz główny (PK), który zapewnia jednoznaczną identyfikację rekordów. Przykładowo:

- Customers(CustomerID)
- Orders(OrderID)
- Products(ProductID)

Dodatkowo w modelu zastosowano tabele asocjacyjne z kluczami głównymi złożonymi, które odwzorowują relacje wiele-do-wielu:

- BillOfMaterials(ProductID, ComponentID)
- ProductionOrderComponents(ProductionOrderID, ComponentID)

---

2. Integralność referencyjna (klucze obce)

Pomiędzy tabelami zdefiniowano klucze obce (**FK**), które zapewniają spójność relacji logicznych w bazie danych. Najważniejsze relacje obejmują:

- Orders.CustomerID → Customers.CustomerID
- Orders.EmployeeID → Employees.EmployeeID
- Orders.ShipperID → Shippers.ShipperID
- OrderDetails.OrderID → Orders.OrderID
- OrderDetails.ProductID → Products.ProductID
- Invoices.OrderID → Orders.OrderID

- InvoiceItems.InvoiceID → Invoices.InvoiceID
- Payments.InvoiceID → Invoices.InvoiceID
- Products.CategoryID → ProductCategories.CategoryID
- Components.ComponentCategoryID → ComponentCategories.ComponentCategoryID
- BillOfMaterials.ProductID → Products.ProductID
- BillOfMaterials.ComponentID → Components.ComponentID
- ProductionOrders.ProductID → Products.ProductID
- ProductionOrders.SourceOrderID → Orders.OrderID
- ProductionOrders.WorkCenterID → WorkCenters.WorkCenterID
- WorkCenterCapacity.WorkCenterID → WorkCenters.WorkCenterID

Ponadto tabela **ComponentReservations** jest powiązana z tabelami **Components**, **OrderDetails** oraz **ProductionOrders**, co umożliwia kontrolę dostępności komponentów na potrzeby realizacji zamówień i produkcji.

---

3. Integralność dziedzinowa (CHECK – ograniczenia wartości)

W schemacie zastosowano ograniczenia **CHECK**, które wymuszają poprawność i logiczną spójność danych. Przykładowe warunki obejmują:

- **Products**
    - UnitPriceNet > 0
    - ProductionTimeInHours > 0
    - ProductionCost > 0
- **ComponentDeliveries**
    - Quantity > 0
- **BillOfMaterials**
    - QuantityPerProduct > 0
- **OrderDetails**
    - UnitPrice > 0
    - Quantity > 0
- **InvoiceItems**
    - Quantity > 0
    - UnitNetPrice > 0

Rezerwacje i stany magazynowe:

- **ProductStocks**
    - QuantityReserved >= 0
- **ComponentStocks**
    - QuantityReserved >= 0
- **ComponentReservations**
    - ReservedQuantity > 0

Produkcja i finanse:

- **ProductionOrders**
    - QuantityToProduce > 0
- **ProductionOrderComponents**
    - QuantityRequired > 0
- **Invoices**
    - TotalNet > 0
    - TotalGross > 0

---

4. Integralność unikalności (UNIQUE / klucze alternatywne)

W bazie danych zdefiniowano ograniczenia unikalności (**UNIQUE / AK**) dla wybranych atrybutów, w celu zapobiegania duplikacji danych:

- **Customers.NIP** – unikalny numer NIP dla klientów biznesowych
- **Invoices.InvoiceNumber** – unikalny numer faktury
- **Deliveries.ParcelNumber** – unikalny numer przesyłki
- **WorkCenters.WorkCenterName** – unikalna nazwa stanowiska / warsztatu

---

Zastosowane warunki integralności zapewniają spójność, poprawność oraz wysoką jakość danych przechowywanych w systemie, a także umożliwiają prawidłową realizację procesów sprzedaży, produkcji i gospodarki magazynowej.

2. Tables

2.1. Table Customers

2.1.1. Columns

Column name	Type	Properties	Description
CustomerID	int	PK	
CompanyName	varchar(100)	null	
ContactName	varchar(60)	null	
ContactTitle	varchar(60)	null	
IsBusiness	bit		
NIP	varchar(15)	null	Unikalny numer NIP
Address	varchar(100)	null	
City	varchar(50)	null	
Region	varchar(50)	null	
PostalCode	varchar(20)	null	
Country	varchar(50)	null	
Phone	varchar(30)	null	
Email	varchar(100)	null	

2.1.2. Alternate keys

Key name	Columns	Description
NIP	NIP	

2.2. Table Departments

2.2.1. Columns

Column name	Type	Properties	Description
DepartmentID	int	PK	
DepartmentName	varchar(100)		
Address	varchar(100)		

2.3. Table Employees

2.3.1. Columns

Column name	Type	Properties	Description
EmployeeID	int	PK	
LastName	varchar(50)	null	
FirstName	varchar(50)	null	
Title	varchar(50)	null	
BirthDate	datetime	null	
HireDate	datetime	null	

Address	varchar(100)	null	
City	varchar(50)	null	
Region	varchar(50)	null	
PostalCode	varchar(20)	null	
Country	varchar(50)	null	
HomePhone	varchar(30)	null	
DepartmentID	int		

2.4. Table Shippers

2.4.1. Columns

Column name	Type	Properties	Description
ShipperID	int	PK	
CompanyName	varchar(100)	null	
Address	varchar(100)	null	
Phone	varchar(30)	null	
ContactEmail	varchar(100)	null	

2.5. Table ProductCategories

2.5.1. Columns

Column name	Type	Properties	Description
CategoryID	int	PK	
CategoryName	varchar(50)		
Description	text	null	

2.6. Table Products

2.6.1. Columns

Column name	Type	Properties	Description
ProductID	int	PK	
ProductName	varchar(100)		
CategoryID	int		
UnitPriceNet	decimal(19,4)		UnitPriceNet > 0
ProductionTimeInHours	int		ProductionTimeInHours > 0
ProductionCost	decimal(19,4)		ProductionCost > 0

2.7. Table ProductStocks

2.7.1. Columns

Column name	Type	Properties	Description
ProductID	int	PK	

QuantityAvailable	decimal(12,2)		
QuantityReserved	decimal(12,2)		QuantityReserved >= 0

2.8. Table ProductStockRegister

2.8.1. Columns

Column name	Type	Properties	Description
ProductStockEntryID	int	PK	
ProductID	int		
EntryDate	datetime		
Amount	decimal(12,2)		

2.9. Table ComponentCategories

2.9.1. Columns

Column name	Type	Properties	Description
ComponentCategoryID	int	PK	
CategoryName	varchar(50)		

2.10. Table Components

2.10.1. Columns

Column name	Type	Properties	Description
ComponentID	int	PK	
ComponentName	varchar(100)		
ComponentCategoryID	int		
UnitOfMeasure	varchar(20)	null	
UnitCost	decimal(19,4)		UnitCost > 0

2.11. Table ComponentStocks

2.11.1. Columns

Column name	Type	Properties	Description
ComponentID	int	PK	
QuantityAvailable	decimal(12,4)		
QuantityReserved	decimal(12,4)	null	QuantityReserved >= 0

2.12. Table ComponentStockRegister

2.12.1. Columns

Column name	Type	Properties	Description
ComponentStockEntryID	int	PK	
ComponentID	int		
EntryDate	datetime	null	
Amount	decimal(12,4)		

2.13. Table ComponentDeliveries

2.13.1. Columns

Column name	Type	Properties	Description
ComponentDeliveryID	int	PK	
ComponentID	int		
Quantity	decimal(12,4)		Quantity > 0
DateOfDelivery	date	null	
Status	varchar(30)		

2.14. Table BillOfMaterials

2.14.1. Columns

Column name	Type	Properties	Description
ProductID	int	PK	
ComponentID	int	PK	
QuantityPerProduct	decimal(12,4)		QuantityPerProduct > 0

2.15. Table OrderStatus

2.15.1. Columns

Column name	Type	Properties	Description
StatusID	int	PK	
StatusCategory	varchar(30)		

2.16. Table Orders

2.16.1. Columns

Column name	Type	Properties	Description
OrderID	int	PK	
CustomerID	int		
EmployeeID	int		

OrderDate	datetime		
RequiredDate	datetime	null	
ShippedDate	datetime	null	
ShipperID	int		
Freight	decimal(19,4)	null	
ShipName	varchar(100)	null	
ShipAddress	varchar(100)	null	
ShipCity	varchar(50)	null	
ShipRegion	varchar(50)	null	
ShipPostalCode	varchar(20)	null	
ShipCountry	varchar(50)	null	
OrderDiscountPercent	decimal(5,2)	null	
StatusID	int		

2.17. Table OrderDetails

2.17.1. Columns

Column name	Type	Properties	Description
OrderDetailID	int	PK	
OrderID	int		
ProductID	int		
UnitPrice	decimal(19,4)		UnitPrice > 0
Quantity	int		Quantity > 0
Discount	decimal(5,2)	null	
Status	varchar(30)		

2.18. Table Complaints

2.18.1. Columns

Column name	Type	Properties	Description
ComplaintID	int	PK	
OrderID	int		
ProductID	int		
EmployeeID	int		
ComplaintDate	datetime		
Status	varchar(30)		

2.19. Table Deliveries

2.19.1. Columns

Column name	Type	Properties	Description
DeliveryID	int	PK	
ParcelNumber	varchar(30)	null	Unikalny numer

			przesyłki
ShippingDate	datetime	null	
EstimatedDeliveryDate	datetime	null	
ShipperID	int		
OrderID	int		
DeliveryStatus	varchar(30)		

2.19.2. Alternate keys

Key name	Columns	Description
AK_0	ParcelNumber	

2.20. Table Invoices

2.20.1. Columns

Column name	Type	Properties	Description
InvoiceID	int	PK	
OrderID	int		
InvoiceNumber	varchar(50)		Unikalny numer faktury
InvoiceDate	datetime		
DueDate	datetime		
TotalNet	decimal(19,4)		TotalNet > 0
TotalVat	decimal(19,4)	null	
TotalGross	decimal(19,4)		TotalGross > 0

2.20.2. Alternate keys

Key name	Columns	Description
InvoiceNumber	InvoiceNumber	

2.21. Table InvoiceItems

2.21.1. Columns

Column name	Type	Properties	Description
InvoiceItemID	int	PK	
InvoiceID	int		
ProductID	int		
Description	varchar(200)		
Quantity	int		Quantity > 0
UnitNetPrice	decimal(19,4)		UnitNetPrice > 0
VatRate	decimal(5,2)	null	
DiscountPercent	decimal(5,2)	null	

2.22. Table Payments

2.22.1. Columns

Column name	Type	Properties	Description
PaymentID	int	PK	
InvoiceID	int		
PaymentDate	datetime	null	
Amount	decimal(19,4)	null	
PaymentMethod	varchar(30)	null	
PaymentStatus	varchar(30)		

2.23. Table ProductionOrders

2.23.1. Columns

Column name	Type	Properties	Description
ProductionOrderID	int	PK	
ProductID	int		
QuantityToProduce	decimal(12,2)		QuantityToProduce > 0
PlannedStartDate	datetime	null	
PlannedEndDate	datetime	null	
SourceOrderID	int		
WorkCenterID	int		
ProductionOrderStatusID	int		

2.24. Table ProductionOrderComponents

2.24.1. Columns

Column name	Type	Properties	Description
ProductionOrderID	int	PK	
ComponentID	int	PK	
QuantityRequired	decimal(12,4)		QuantityRequired > 0
QuantityIssued	decimal(12,4)		

2.25. Table WorkCenters

Description:

- do modelu mocy produkcyjnych

2.25.1. Columns

Column name	Type	Properties	Description
WorkCenterID	int	PK	

WorkCenterName	varchar(80)		- unikalna nazwa zakładu pracy/warsztatu
IsActive	bit		

2.25.2. Alternate keys

Key name	Columns	Description
UQ_WorkCenters_Name	WorkCenterName	

2.26. Table WorkCenterCapacity

Description:

- moc przerobowa/produkcyjna w jednostce czasu

2.26.1. Columns

Column name	Type	Properties	Description
WorkCenterID	int	PK	
CapacityHoursPerDay	float		

2.27. Table ProductionOrderStatus

Description:

- Słownik statusów zleceń produkcyjnych

2.27.1. Columns

Column name	Type	Properties	Description
ProductionOrderStatusID	int	PK	
StatusName	varchar(30)		

2.28. Table ComponentReservations

Description:

- „System powinien umożliwiać kontrolę dostępności materiałów / komponentów na potrzeby realizacji zamówień oraz produkcji.”

2.28.1. Columns

Column name	Type	Properties	Description
ComponentReservationID	int	PK	
ComponentID	int		
OrderDetailID	int		
ProductionOrderI	int		

D			
ReservedQuantity	decimal(12,4)		ReservedQuantity > 0

2.29. Table WorkingCostParameters

Description:

- koszt produkcji danego elementu za godzinę

2.29.1. Columns

Column name	Type	Properties	Description
CostParametersID	int	PK	
LabourRatePerHour	money		

3. References

3.1. Reference FK_0

Departments	0..*	Employees
DepartmentID	<->	DepartmentID

3.2. Reference FK_1

ProductCategories	0..*	Products
CategoryID	<->	CategoryID

3.3. Reference FK_2

Products	0..1	ProductStocks
ProductID	<->	ProductID

3.4. Reference FK_3

Products	0..*	ProductStockRegister
ProductID	<->	ProductID

3.5. Reference FK_4

ComponentCategories	0..*	Components
ComponentCategoryID	<->	ComponentCategoryID

3.6. Reference FK_5

Components	0..1	ComponentStocks
ComponentID	<->	ComponentID

3.7. Reference FK_6

Components	0..*	ComponentStockRegister
ComponentID	<->	ComponentID

3.8. Reference FK_7

Components	0..*	ComponentDeliveries
ComponentID	<->	ComponentID

3.9. Reference FK_8

Products	0..*	BillOfMaterials
ProductID	<->	ProductID

3.10. Reference FK_9

Components	0..*	BillOfMaterials
ComponentID	<->	ComponentID

3.11. Reference FK_10

Customers	0..*	Orders
CustomerID	<->	CustomerID

3.12. Reference FK_11

Employees	0..*	Orders
EmployeeID	<->	EmployeeID

3.13. Reference FK_12

Shippers	0..*	Orders
ShipperID	<->	ShipperID

3.14. Reference FK_13

OrderStatus	0..*	Orders
StatusID	<->	StatusID

3.15. Reference FK_14

Orders	0..*	OrderDetails
OrderID	<->	OrderID

3.16. Reference FK_15

Products	0..*	OrderDetails
ProductID	<->	ProductID

3.17. Reference FK_16

Orders	0..*	Complaints
OrderID	<->	OrderID

3.18. Reference FK_17

Products	0..*	Complaints
ProductID	<->	ProductID

3.19. Reference FK_18

Employees	0..*	Complaints
EmployeeID	<->	EmployeeID

3.20. Reference FK_19

Shippers	0..*	Deliveries
ShipperID	<->	ShipperID

3.21. Reference FK_20

Orders	0..*	Deliveries
OrderID	<->	OrderID

3.22. Reference FK_21

Orders	0..*	Invoices
OrderID	<->	OrderID

3.23. Reference FK_22

Invoices	0..*	InvoiceItems
InvoiceID	<->	InvoiceID

3.24. Reference FK_23

Products	0..*	InvoiceItems
ProductID	<->	ProductID

3.25. Reference FK_24

Invoices	0..*	Payments
InvoiceID	<->	InvoiceID

3.26. Reference FK_25

Products	0..*	ProductionOrders
ProductID	<->	ProductID

3.27. Reference FK_26

Orders	0..*	ProductionOrders
OrderID	<->	SourceOrderID

3.28. Reference FK_27

ProductionOrders	0..*	ProductionOrderComponents
ProductionOrderID	<->	ProductionOrderID

3.29. Reference FK_28

Components	0..*	ProductionOrderComponents
ComponentID	<->	ComponentID

3.30. Reference WorkCenterCapacity_WorkCenters

WorkCenters	0..*	WorkCenterCapacity
WorkCenterID	<->	WorkCenterID

3.31. Reference ProductionOrders_WorkCenters

WorkCenters	0..*	ProductionOrders
WorkCenterID	<->	WorkCenterID

3.32. Reference ProductionOrders_ProductionOrderStatus

ProductionOrderStatus	0..*	ProductionOrders
ProductionOrderStatusID	<->	ProductionOrderStatusID

3.33. Reference ComponentReservations_Components

Components	0..*	ComponentReservations
ComponentID	<->	ComponentID

3.34. Reference ComponentReservations_OrderDetails

OrderDetails	0..*	ComponentReservations
OrderDetailID	<->	OrderDetailID

3.35. Reference ComponentReservations_ProductionOrders

ProductionOrders	0..*	ComponentReservations
ProductionOrderID	<->	ProductionOrderID

Widoki

Poniżej przedstawiono zestaw widoków utworzonych w bazie danych.

Widoki służą do generowania raportów oraz zestawień wymaganych w opisie projektu.

Ułatwiają one analizę danych bez konieczności pisania złożonych zapytań SQL.

1. Widok vw_OrderLinesNet

Wyświetla szczegółowe informacje o pozycjach zamówień wraz z naliczonymi rabatami. Widok oblicza wartości brutto oraz końcowe wartości netto każdej pozycji zamówienia.

```
CREATE OR ALTER VIEW dbo.vw_OrderLinesNet
AS
SELECT
    o.OrderID,
    o.CustomerID,
    o.EmployeeID,
    o.OrderDate,
    o.RequiredDate,
    o.ShippedDate,
    o.StatusID,
    os.StatusCategory,
    od.OrderDetailID,
    od.ProductID,
    p.ProductName,
    p.CategoryID,
    pc.CategoryName,
    od.Quantity,
    od.UnitPrice,

    -- Rabat pozycji (OrderDetails.Discount) i rabat zamówienia
(Orders.OrderDiscountPercent)
    ISNULL(od.Discount, 0) AS LineDiscountPercent,
    ISNULL(o.OrderDiscountPercent, 0) AS OrderDiscountPercent,

    -- Brutto: ilość * cena
    CAST(od.Quantity * od.UnitPrice AS decimal(19,4)) AS LineValueGross,
    -- Po rabacie pozycji
    CAST(
        od.Quantity * od.UnitPrice * (1 - ISNULL(od.Discount,0)/100.0)
        AS decimal(19,4)
    ) AS LineValueAfterLineDiscount,
    -- Netto finalne: rabat pozycji + rabat zamówienia
    CAST(
        od.Quantity * od.UnitPrice
        * (1 - ISNULL(od.Discount,0)/100.0)
        * (1 - ISNULL(o.OrderDiscountPercent,0)/100.0)
        AS decimal(19,4)
    ) AS LineValueNet
FROM dbo.Orders o
JOIN dbo.OrderDetails od      ON od.OrderID = o.OrderID
JOIN dbo.Products p           ON p.ProductID = od.ProductID
JOIN dbo.ProductCategories pc ON pc.CategoryID = p.CategoryID
JOIN dbo.OrderStatus os       ON os.StatusID = o.StatusID;
GO
```

2. Widok vw_SalesByCategory_Weekly

Wyświetla tygodniowe statystyki sprzedaży z podziałem na kategorie produktów. Pokazuje liczbę zamówień, ilość sprzedanych produktów oraz przychód netto.

```
CREATE OR ALTER VIEW dbo.vw_SalesByCategory_Weekly
AS
SELECT
    DATEPART(ISO_YEAR, oln.OrderDate) AS ISOYear,
    DATEPART(ISO_WEEK, oln.OrderDate) AS ISOWeek,
    oln.CategoryID,
    oln.CategoryName,
    COUNT(DISTINCT oln.OrderID) AS OrdersCount,
    SUM(CAST(oln.Quantity AS decimal(19,4))) AS QuantitySold,
    SUM(oln.LineValueNet) AS RevenueNet
FROM dbo.vw_OrderLinesNet oln
GROUP BY
    DATEPART(ISO_YEAR, oln.OrderDate),
    DATEPART(ISO_WEEK, oln.OrderDate),
    oln.CategoryID,
    oln.CategoryName;
GO
```

3. Widok vw_SalesByCategory_Monthly

Wyświetla miesięczne statystyki sprzedaży z podziałem na kategorie produktów. Umożliwia analizę przychodów i ilości sprzedaży w ujęciu miesięcznym.

```
CREATE OR ALTER VIEW dbo.vw_SalesByCategory_Monthly
AS
SELECT
    YEAR(oln.OrderDate) AS [Year],
    MONTH(oln.OrderDate) AS [Month],
    oln.CategoryID,
    oln.CategoryName,
    COUNT(DISTINCT oln.OrderID) AS OrdersCount,
    SUM(CAST(oln.Quantity AS decimal(19,4))) AS QuantitySold,
    SUM(oln.LineValueNet) AS RevenueNet
FROM dbo.vw_OrderLinesNet oln
GROUP BY
    YEAR(oln.OrderDate),
    MONTH(oln.OrderDate),
    oln.CategoryID,
    oln.CategoryName;
GO
```

4. Widok vw_UnitProductionCost_PerProduct

Wyświetla jednostkowy koszt produkcji oraz czas produkcji dla każdego produktu. Widok pozwala analizować koszty wytwarzania pojedynczych produktów.

```
CREATE OR ALTER VIEW dbo.vw_UnitProductionCost_PerProduct
AS
SELECT
    p.ProductID,
    p.ProductName,
    p.CategoryID,
    pc.CategoryName,
    p.UnitPriceNet,
    p.ProductionTimeInHours,
    p.ProductionCost AS UnitProductionCost
FROM dbo.Products p
JOIN dbo.ProductCategories pc ON pc.CategoryID = p.CategoryID;
GO
```

5. Widok vw_ProductionCostByCategory_Yearly

Wyświetla roczne koszty planowanej produkcji z podziałem na kategorie produktów. Koszt liczony jest na podstawie ilości planowanej produkcji i kosztu jednostkowego.

```
CREATE OR ALTER VIEW dbo.vw_ProductionCostByCategory_Yearly
AS
SELECT
    YEAR(po.PlannedStartDate) AS [Year],
    pc.CategoryID,
    pc.CategoryName,
    SUM(CAST(po.QuantityToProduce AS decimal(19,4))) AS QtyPlanned,
    SUM(CAST(po.QuantityToProduce AS decimal(19,4)) * CAST(p.ProductionCost AS
decimal(19,4))) AS PlannedProductionCost
FROM dbo.ProductionOrders po
JOIN dbo.Products p           ON p.ProductID = po.ProductID
JOIN dbo.ProductCategories pc ON pc.CategoryID = p.CategoryID
GROUP BY
    YEAR(po.PlannedStartDate),
    pc.CategoryID,
    pc.CategoryName;
GO
```

6. Widok vw_ProductionCostByCategory_Quarterly

Wyświetla kwartalne koszty planowanej produkcji dla poszczególnych kategorii produktów. Umożliwia analizę kosztów produkcji w ujęciu kwartalnym.

```
CREATE OR ALTER VIEW dbo.vw_ProductionCostByCategory_Quarterly
AS
SELECT
    YEAR(po.PlannedStartDate) AS [Year],
    DATEPART(QUARTER, po.PlannedStartDate) AS [Quarter],
    pc.CategoryID,
    pc.CategoryName,
    SUM(CAST(po.QuantityToProduce AS decimal(19,4))) AS QtyPlanned,
    SUM(CAST(po.QuantityToProduce AS decimal(19,4)) * CAST(p.ProductionCost AS
decimal(19,4))) AS PlannedProductionCost
FROM dbo.ProductionOrders po
JOIN dbo.Products p           ON p.ProductID = po.ProductID
JOIN dbo.ProductCategories pc ON pc.CategoryID = p.CategoryID
GROUP BY
    YEAR(po.PlannedStartDate),
    DATEPART(QUARTER, po.PlannedStartDate),
    pc.CategoryID,
    pc.CategoryName;
GO
```

7. Widok vw_ProductionCostByCategory_Monthly

Wyświetla miesięczne koszty planowanej produkcji z podziałem na kategorie produktów. Pozwala śledzić zmiany kosztów produkcji w czasie.

```
CREATE OR ALTER VIEW dbo.vw_ProductionCostByCategory_Monthly
AS
SELECT
    YEAR(po.PlannedStartDate) AS [Year],
    MONTH(po.PlannedStartDate) AS [Month],
    pc.CategoryID,
    pc.CategoryName,
    SUM(CAST(po.QuantityToProduce AS decimal(19,4))) AS QtyPlanned,
    SUM(CAST(po.QuantityToProduce AS decimal(19,4)) * CAST(p.ProductionCost AS
decimal(19,4))) AS PlannedProductionCost
FROM dbo.ProductionOrders po
JOIN dbo.Products p           ON p.ProductID = po.ProductID
JOIN dbo.ProductCategories pc ON pc.CategoryID = p.CategoryID
GROUP BY
    YEAR(po.PlannedStartDate),
    MONTH(po.PlannedStartDate),
    pc.CategoryID,
    pc.CategoryName;
GO
```

8. Widok vw_ProductionCost_Weekly

Wyświetla tygodniowe koszty planowanej produkcji dla całej firmy. Widok służy do szybkiej analizy kosztów produkcji w krótkich okresach czasu.

```
CREATE OR ALTER VIEW dbo.vw_ProductionCost_Weekly
AS
SELECT
    DATEPART(ISO_YEAR, po.PlannedStartDate) AS ISOYear,
    DATEPART(ISO_WEEK, po.PlannedStartDate) AS ISOWeek,
    SUM(CAST(po.QuantityToProduce AS decimal(19,4)) * CAST(p.ProductionCost AS
decimal(19,4))) AS PlannedProductionCost
FROM dbo.ProductionOrders po
JOIN dbo.Products p ON p.ProductID = po.ProductID
GROUP BY
    DATEPART(ISO_YEAR, po.PlannedStartDate),
    DATEPART(ISO_WEEK, po.PlannedStartDate);
GO
```

9. Widok vw_ProductionCost_Monthly

Wyświetla miesięczne koszty planowanej produkcji. Umożliwia porównywanie kosztów produkcji pomiędzy kolejnymi miesiącami.

```
CREATE OR ALTER VIEW dbo.vw_ProductionCost_Monthly
AS
SELECT
    YEAR(po.PlannedStartDate) AS [Year],
    MONTH(po.PlannedStartDate) AS [Month],
    SUM(CAST(po.QuantityToProduce AS decimal(19,4)) * CAST(p.ProductionCost AS
decimal(19,4))) AS PlannedProductionCost
FROM dbo.ProductionOrders po
JOIN dbo.Products p ON p.ProductID = po.ProductID
GROUP BY
    YEAR(po.PlannedStartDate),
    MONTH(po.PlannedStartDate);
GO
```

10. Widok vw_CurrentProductStock

Wyświetla aktualne stany magazynowe produktów. Pokazuje ilość dostępną, zarezerwowaną oraz faktycznie wolną w magazynie.

```
CREATE OR ALTER VIEW dbo.vw_CurrentProductStock
AS
SELECT
    ps.ProductID,
    p.ProductName,
    p.CategoryID,
    pc.CategoryName,
    ps.QuantityAvailable,
    ps.QuantityReserved,
    CAST(ps.QuantityAvailable - ps.QuantityReserved AS decimal(12,2)) AS
QuantityFree
FROM dbo.ProductStocks ps
JOIN dbo.Products p           ON p.ProductID = ps.ProductID
JOIN dbo.ProductCategories pc ON pc.CategoryID = p.CategoryID;
GO
```

11. Widok vw_PlannedProduction_Products

Wyświetla produkty zaplanowane do produkcji wraz z terminami oraz statusem zleceń. Widok służy do przeglądania aktualnego planu produkcji.

```
CREATE OR ALTER VIEW dbo.vw_PlannedProduction_Products
AS
SELECT
    po.ProductionOrderID,
    po.ProductID,
    p.ProductName,
    p.CategoryID,
    pc.CategoryName,
    po.QuantityToProduce,
    po.PlannedStartDate,
    po.PlannedEndDate,
    po.SourceOrderID,
    po.WorkCenterID,
    wc.WorkCenterName,
    po.ProductionOrderStatusID,
    pos.StatusName AS ProductionStatus
FROM dbo.ProductionOrders po
JOIN dbo.Products p               ON p.ProductID = po.ProductID
JOIN dbo.ProductCategories pc     ON pc.CategoryID = p.CategoryID
JOIN dbo.WorkCenters wc           ON wc.WorkCenterID = po.WorkCenterID
JOIN dbo.ProductionOrderStatus pos ON pos.ProductionOrderStatusID =
po.ProductionOrderStatusID;
GO
```

12. Widok vw_ProductionPlan_WithLoad

Wyświetla plan produkcji wraz z obciążeniem stanowisk roboczych. Pokazuje wymagany czas produkcji oraz dostępne moce przerobowe.

```
CREATE OR ALTER VIEW dbo.vw_ProductionPlan_WithLoad
AS
SELECT
    po.ProductionOrderID,
    po.ProductID,
    p.ProductName,
    po.QuantityToProduce,
    po.PlannedStartDate,
    po.PlannedEndDate,
    po.WorkCenterID,
    wc.WorkCenterName,
    cap.CapacityHoursPerDay,
    CAST(CAST(po.QuantityToProduce AS decimal(19,4)) *
CAST(p.ProductionTimeInHours AS decimal(19,4)) AS decimal(19,4)) AS RequiredHours
FROM dbo.ProductionOrders po
JOIN dbo.Products p            ON p.ProductID = po.ProductID
JOIN dbo.WorkCenters wc        ON wc.WorkCenterID = po.WorkCenterID
JOIN dbo.WorkCenterCapacity cap ON cap.WorkCenterID = po.WorkCenterID;
GO
```

13. Widok vw_CustomerOrderHistory_WithDiscounts

Wyświetla historię zamówień klientów wraz z zastosowanymi rabatami. Widok umożliwia analizę zakupów klientów w różnych przedziałach czasowych.

```
CREATE OR ALTER VIEW dbo.vw_CustomerOrderHistory_WithDiscounts
AS
SELECT
    c.CustomerID,
    c.CompanyName,
    c.ContactName,
    c.IsB2B,
    c.NIP,

    o.OrderID,
    o.OrderDate,
    os.StatusCategory AS OrderStatusCategory,
    ISNULL(o.OrderDiscountPercent,0) AS OrderDiscountPercent,

    od.OrderDetailID,
    od.ProductID,
    p.ProductName,
    pc.CategoryName,

    od.Quantity,
    od.UnitPrice,
    ISNULL(od.Discount,0) AS LineDiscountPercent,

    CAST(od.Quantity * od.UnitPrice AS decimal(19,4)) AS LineValueGross,
    CAST(od.Quantity * od.UnitPrice * (1 - ISNULL(od.Discount,0)/100.0) AS
decimal(19,4)) AS LineValueAfterLineDiscount,
    CAST(
        od.Quantity * od.UnitPrice
        * (1 - ISNULL(od.Discount,0)/100.0)
        * (1 - ISNULL(o.OrderDiscountPercent,0)/100.0)
        AS decimal(19,4)
    ) AS LineValueNetFinal
FROM dbo.Customers c
JOIN dbo.Orders o             ON o.CustomerID = c.CustomerID
JOIN dbo.OrderStatus os       ON os.StatusID  = o.StatusID
JOIN dbo.OrderDetails od      ON od.OrderID   = o.OrderID
JOIN dbo.Products p           ON p.ProductID  = od.ProductID
JOIN dbo.ProductCategories pc ON pc.CategoryID = p.CategoryID;
GO
```

14. Widok vw_CurrentComponentStock

Wyświetla aktualne stany magazynowe komponentów wykorzystywanych w produkcji. Pokazuje ilość dostępną, zarezerwowaną oraz wolną dla każdego komponentu.

```
CREATE OR ALTER VIEW dbo.vw_CurrentComponentStock
AS
SELECT
    cs.ComponentID,
    c.ComponentName,
    c.ComponentCategoryID,
    cc.CategoryName AS ComponentCategoryName,
    c.UnitOfMeasure,
    c.UnitCost,
    cs.QuantityAvailable,
    cs.QuantityReserved,
    CAST(cs.QuantityAvailable - cs.QuantityReserved AS decimal(12,4)) AS
QuantityFree
FROM dbo.ComponentStocks cs
JOIN dbo.Components c           ON c.ComponentID = cs.ComponentID
JOIN dbo.ComponentCategories cc ON cc.ComponentCategoryID = c.ComponentCategoryID;
GO
```

15. Widok vw_ComponentDemand_ForPlannedProduction

Wyświetla zapotrzebowanie na komponenty wynikające z planu produkcji. Widok wspiera planowanie materiałowe (MRP) i kontrolę dostępności komponentów.

```
CREATE OR ALTER VIEW dbo.vw_ComponentDemand_ForPlannedProduction
AS
SELECT
    po.ProductionOrderID,
    po.PlannedStartDate,
    po.PlannedEndDate,
    po.ProductID,
    p.ProductName,

    bom.ComponentID,
    c.ComponentName,
    c.UnitOfMeasure,

    CAST(CAST(po.QuantityToProduce AS decimal(19,4)) * CAST(bom.QuantityPerProduct
AS decimal(19,4)) AS decimal(19,4)) AS ComponentQtyRequired
FROM dbo.ProductionOrders po
JOIN dbo.Products p          ON p.ProductID = po.ProductID
JOIN dbo.BillOfMaterials bom ON bom.ProductID = po.ProductID
JOIN dbo.Components c        ON c.ComponentID = bom.ComponentID;
GO
```

Procedury

Poniżej przedstawiono zestaw procedur zaprojektowanych w bazie danych. Służą one do dodawaia nowych elementów do tabel, aktualizacji wartości pól w tabelach oraz obsługi logiki biznesowej bazy.

AddComponentToProduct

Procedura dodaje nowy składnik do produktu w tabeli BillOfMaterials po walidacji.

```
CREATE OR ALTER PROCEDURE [dbo].[sp_DodajSkladnikDoProduktu]
    @ProductID      INT,            -- Produkt do którego dodajemy komponent?
    @ComponentID    INT,            -- Dodawany komponent
    @Quantity       DECIMAL(12,4)   -- Ile komponentu potrzeba na 1 produkt
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRANSACTION;

        -- 1. Walidacja istnienia
        IF NOT EXISTS (SELECT 1 FROM [dbo].[Products] WHERE ProductID =
@ProductID)
            THROW 51049, 'Błąd: Produkt o podanym ID nie istnieje.', 1;

        IF NOT EXISTS (SELECT 1 FROM [dbo].[Components] WHERE ComponentID =
@ComponentID)
            THROW 51050, 'Błąd: Komponent o podanym ID nie istnieje.', 1;

        -- 2. Walidacja ilości
        IF @Quantity <= 0
            THROW 51051, 'Błąd: Ilość składnika musi być większa od zera.', 1;

        -- 3. Sprawdzenie czy ten składnik już nie jest przypisany
        IF EXISTS (SELECT 1 FROM [dbo].[BillOfMaterials] WHERE ProductID =
@ProductID AND ComponentID = @ComponentID)
        BEGIN
             THROW 51052, 'Błąd: Ten komponent jest już przypisany do tego
produktu.', 1;
        END

        -- 4. Dodanie wpisu do BillOfMaterials
        INSERT INTO [dbo].[BillOfMaterials] (
            [ProductID],
            [ComponentID],
            [QuantityPerProduct]
        )
        VALUES (
            @ProductID,
            @ComponentID,
            @Quantity
```

```
        );

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
GO
```

AddDepartment

Procedura dodaje nowy dział do tabeli Departments i zwraca jego ID po walidacji.

```
CREATE OR ALTER PROCEDURE [dbo].[AddDepartment]
    @DepartmentName VARCHAR(100),       -- Nazwa działu
    @Address        VARCHAR(100) = NULL,-- Adres działu
    @NewDepartmentID INT OUTPUT         -- Zwracamy ID nowego działu
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- 1. Walidacja danych wejściowych

        -- Nazwa działu nie może być pusta
        IF @DepartmentName IS NULL OR LTRIM(RTRIM(@DepartmentName)) = ''
        BEGIN
            THROW 51020, 'Błąd: Nazwa działu (DepartmentName) jest wymagana.', 1;
        END

        -- 2. Sprawdzenie unikalności
        IF EXISTS (SELECT 1 FROM [dbo].[Departments] WHERE DepartmentName =
@DepartmentName)
        BEGIN
            THROW 51021, 'Błąd: Dział o podanej nazwie już istnieje w strukturze
firmy.', 1;
        END

        -- 3. Wstawienie rekordu
        INSERT INTO [dbo].[Departments] (
            [DepartmentName],
            [Address]
        )
        VALUES (
            @DepartmentName,
            @Address
        );
```

```
        -- 4. Pobranie ID nowo dodanego działu
        SET @NewDepartmentID = SCOPE_IDENTITY();

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
GO
```

AddEmployee

Procedura dodaje nowego pracownika do tabeli Employees po walidacji danych i zwraca jego ID.

```
CREATE OR ALTER PROCEDURE [dbo].[sp_DodajPracownika]
    @LastName       VARCHAR(50),          -- Nazwisko (wymagane)
    @FirstName      VARCHAR(50),          -- Imię (wymagane)
    @Title          VARCHAR(50) = NULL,   -- Stanowisko
    @BirthDate      DATETIME = NULL,      -- Data urodzenia
    @HireDate       DATETIME = NULL,      -- Data zatrudnienia
    @Address        VARCHAR(100) = NULL,
    @City           VARCHAR(50) = NULL,
    @Region         VARCHAR(50) = NULL,
    @PostalCode     VARCHAR(20) = NULL,
    @Country        VARCHAR(50) = NULL,
    @HomePhone      VARCHAR(30) = NULL,
    @DepartmentID   INT,                  -- Dział (wymagane)
    @NewEmployeeID  INT OUTPUT            -- Zwracane ID pracownika
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- 1. WALIDACJA DANYCH

        -- Sprawdzenie imienia i nazwiska
        IF @LastName IS NULL OR @FirstName IS NULL
        BEGIN
            THROW 51022, 'Błąd: Imię i Nazwisko pracownika są wymagane.', 1;
        END

        -- Sprawdzenie czy podany dział istnieje
        IF NOT EXISTS (SELECT 1 FROM [dbo].[Departments] WHERE DepartmentID =
@DepartmentID)
        BEGIN
```

```
            THROW 51023, 'Błąd: Podany dział (DepartmentID) nie istnieje.', 1;
        END

        -- Logika dat: Data zatrudnienia nie może być wcześniejsza niż urodzenia
        IF @BirthDate IS NOT NULL AND @HireDate IS NOT NULL
        BEGIN
            IF @HireDate < @BirthDate
            BEGIN
                THROW 51024, 'Błąd: Data zatrudnienia nie może być wcześniejsza
niż data urodzenia.', 1;
            END
        END

        -- Domyślna data zatrudnienia = GETDATE() (jeśli nie podano)
        IF @HireDate IS NULL
            SET @HireDate = GETDATE();

        -- 2. WSTAWIENIE PRACOWNIKA
        INSERT INTO [dbo].[Employees] (
            [LastName],
            [FirstName],
            [Title],
            [BirthDate],
            [HireDate],
            [Address],
            [City],
            [Region],
            [PostalCode],
            [Country],
            [HomePhone],
            [DepartmentID]
        )
        VALUES (
            @LastName,
            @FirstName,
            @Title,
            @BirthDate,
            @HireDate,
            @Address,
            @City,
            @Region,
            @PostalCode,
            @Country,
            @HomePhone,
            @DepartmentID
        );

        -- 3. POBRANIE ID
        SET @NewEmployeeID = SCOPE_IDENTITY();

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
```

```
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
GO
```

AddPositionToOrder

Procedura dodaje nową pozycję do zamówienia, rezerwuje dostępny towar i ustala status pozycji, zwracając jej ID.

```
CREATE OR ALTER PROCEDURE [dbo].[AddPositionToOrder]
    @OrderID        INT,                -- Zamowienie, do którego dodajemy pozycję
    @ProductID      INT,                -- Dodawany produkt
    @Quantity       INT,                -- Liczba sztuk
    @Discount       DECIMAL(5,2) = 0,   -- Rabat na konkretną pozycję
    @NewDetailID    INT OUTPUT          -- Zwracane ID nowej pozycji
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- 1. WALIDACJA PODSTAWOWA
        IF NOT EXISTS (SELECT 1 FROM [dbo].[Orders] WHERE OrderID = @OrderID)
            THROW 51012, 'Błąd: Zamówienie o podanym ID nie istnieje.', 1;

        IF NOT EXISTS (SELECT 1 FROM [dbo].[Products] WHERE ProductID =
@ProductID)
            THROW 51013, 'Błąd: Produkt o podanym ID nie istnieje.', 1;

        IF @Quantity <= 0
            THROW 51014, 'Błąd: Ilość (Quantity) musi być większa od zera.', 1;

        -- 2. POBRANIE CENY PRODUKTU
        -- Cena w zamówieniu wynika z aktualnej ceny produktu (zabezpieczenie
przed zmianami cen w przyszłości)
        DECLARE @CurrentUnitPrice DECIMAL(19,4);

        SELECT @CurrentUnitPrice = UnitPriceNet
        FROM [dbo].[Products]
        WHERE ProductID = @ProductID;

        -- 3. LOGIKA MAGAZYNOWA I USTALENIE STATUSU
        DECLARE @AvailableQty DECIMAL(12,2);
        DECLARE @StatusPozycji VARCHAR(30);

        -- Sprawdzenie stanu magazynoego w tabeli ProductStocks
        SELECT @AvailableQty = QuantityAvailable
        FROM [dbo].[ProductStocks]
```

```
        WHERE ProductID = @ProductID;

        -- Jeśli nie ma rekordu w magazynie, to dostępna ilość = 0
        SET @AvailableQty = ISNULL(@AvailableQty, 0);

        IF @AvailableQty >= @Quantity
        BEGIN
            -- SCENARIUSZ A: JEST TOWAR
            -- Dokonanie rezerwacji towaru
            UPDATE [dbo].[ProductStocks]
            SET QuantityAvailable = QuantityAvailable - @Quantity,
                QuantityReserved = ISNULL(QuantityReserved, 0) + @Quantity
            WHERE ProductID = @ProductID;

            SET @StatusPozycji = 'Zarezerwowano';
        END        ELSE
        BEGIN
            -- SCENARIUSZ B: BRAK TOWARU (lub za mało)
            -- Oznaczenie pozycję jako wymagającą produkcji.

            SET @StatusPozycji = 'Do Produkcji';
        END

        -- 4. WSTAWIENIE POZYCJI ZAMÓWIENIA
        INSERT INTO [dbo].[OrderDetails] (
            [OrderID],
            [ProductID],
            [UnitPrice],    -- Cena pobrana z tabeli Products
            [Quantity],
            [Discount],
            [Status]        -- Status wynikający z logiki magazynowej
        )
        VALUES (
            @OrderID,
            @ProductID,
            @CurrentUnitPrice,
            @Quantity,
            @Discount,
            @StatusPozycji
        );

        -- 5. Pobranie ID nowej pozycji
        SET @NewDetailID = SCOPE_IDENTITY();

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
GO
```

AddProductCategory

Procedura dodaje nową kategorię produktu do tabeli ProductCategories po walidacji i zwraca jej ID.

```
CREATE OR ALTER PROCEDURE [dbo].[sp_DodajKategorieProduktu]
    @CategoryName   VARCHAR(50),         -- Nazwa kategorii (Wymagana)
    @Description    VARCHAR(MAX) = NULL, -- Opis kategorii
    @VATPercentage  DECIMAL(5,2) = 23.00,-- Domyślny VAT dla tej kategorii (np.
23.00)
    @NewCategoryID  INT OUTPUT           -- Zwracane ID nowej kategorii
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- 1. WALIDACJA DANYCH

        -- Nazwa kategorii  (wymagana)
        IF @CategoryName IS NULL OR LTRIM(RTRIM(@CategoryName)) = ''
        BEGIN
            THROW 51029, 'Błąd: Nazwa kategorii (CategoryName) jest wymagana.', 1;
        END

        -- Walidacja stawki VAT (musi być logiczna, (0-100%)
        IF @VATPercentage < 0.00 OR @VATPercentage > 100.00
        BEGIN
            THROW 51030, 'Błąd: Stawka VAT musi mieścić się w przedziale 0-100.',
1;
        END

        -- 2. SPRAWDZENIE DUPLIKATÓW
        IF EXISTS (SELECT 1 FROM [dbo].[ProductCategories] WHERE CategoryName =
@CategoryName)
        BEGIN
            THROW 51031, 'Błąd: Kategoria o podanej nazwie już istnieje.', 1;
        END

        -- 3. WSTAWIENIE REKORDU
        INSERT INTO [dbo].[ProductCategories] (
            [CategoryName],
            [Description],
            [VATPercentage]
        )
        VALUES (
            @CategoryName,
            @Description,
            @VATPercentage
        );
```

```
        -- 4. POBRANIE ID
        SET @NewCategoryID = SCOPE_IDENTITY();

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
GO
```

CompleteProductionOrder

Procedura finalizuje zlecenie produkcyjne: zmienia jego status na „Zakończone”, zużywa komponenty z magazynu, usuwa ich rezerwacje, dodaje wyprodukowany produkt do stanu magazynowego i rejestruje wszystkie zmiany w historii magazynowej.

```
/*
 Finalizacja zlecenia produkcyjnego.
 Po zakończeniu produkcji procedura
 - Zmienia status w ProductionOrders na "Zakończone".
 - Zdejmuje rezerwację z komponentów i trwale usunąć je ze stanu
(ComponentStocks).
 - Dodać wyprodukowany produkt do magazynu (ProductStocks -> QuantityAvailable).
 - Dodać wpis do historii zmian magazynowych (ProductStockRegister).
 */

CREATE OR ALTER PROCEDURE CompleteProductionOrder
    @ProductionOrderID int
AS
BEGIN
    SET NOCOUNT ON;

    -- Deklaracje zmiennych
    DECLARE @ProductID int;
    DECLARE @QtyProduced decimal(12,2);
    DECLARE @CurrentStatusID int;
    DECLARE @CompletedStatusID int;
    DECLARE @WorkCenterID int;

    -- 1. Sprawdzenie czy zlecenie istnieje
    SELECT
        @ProductID = ProductID,
        @QtyProduced = QuantityToProduce,
        @CurrentStatusID = ProductionOrderStatusID,
        @WorkCenterID = WorkCenterID
    FROM ProductionOrders
    WHERE ProductionOrderID = @ProductionOrderID;
```

```
    IF @ProductID IS NULL
    BEGIN
        PRINT 'Błąd: Nie znaleziono zlecenia produkcyjnego o ID: ' +
CAST(@ProductionOrderID AS VARCHAR);
        RETURN;
    END

    -- Pobranie ID statusu 'Zakończone'
    SELECT TOP 1 @CompletedStatusID = ProductionOrderStatusID
    FROM ProductionOrderStatus
    WHERE StatusName LIKE 'Zakończone%' OR StatusName LIKE 'Wykonane%';

    -- Zabezpieczenie na wypadek braku statusu w słowniku
    IF @CompletedStatusID IS NULL SET @CompletedStatusID = 3; -- 3 -> domyślne
zakończone

    IF @CurrentStatusID = @CompletedStatusID
    BEGIN
        PRINT 'Informacja: To zlecenie jest już zakończone.';
        RETURN;
    END

    BEGIN TRANSACTION;

    BEGIN TRY
        -- 2. Aktualizacja statusu zlecenia
        UPDATE ProductionOrders
        SET ProductionOrderStatusID = @CompletedStatusID
        WHERE ProductionOrderID = @ProductionOrderID;

        -- 3. (Zużycie materiałów)
        -- A. Rejestracja historii magazynowej dla komponentów (Wartości ujemne -
zużycie)
        INSERT INTO ComponentStockRegister (ComponentID, EntryDate, Amount)
        SELECT
            ComponentID,
            GETDATE(),
            -QuantityRequired
        FROM ProductionOrderComponents
        WHERE ProductionOrderID = @ProductionOrderID;

        -- B. Aktualizacja stanów magazynowych
        UPDATE S
        SET
            S.QuantityAvailable = S.QuantityAvailable - POC.QuantityRequired,
            S.QuantityReserved = S.QuantityReserved - POC.QuantityRequired
        FROM ComponentStocks S
        INNER JOIN ProductionOrderComponents POC ON S.ComponentID =
POC.ComponentID
        WHERE POC.ProductionOrderID = @ProductionOrderID;

        -- C. Usunięcie rezerwacji z tabeli ComponentReservations
        DELETE FROM ComponentReservations
        WHERE ProductionOrderID = @ProductionOrderID;
```

```
        -- D. Zaktualizowanie ilości wydanej w zleceniu (QuantityIssued)
        UPDATE ProductionOrderComponents
        SET QuantityIssued = QuantityRequired
        WHERE ProductionOrderID = @ProductionOrderID;

        -- 4. Przyjęcie wyrobu gotowego na magazyn

        -- A. Aktualizacja stanu ProductStocks
        IF EXISTS (SELECT 1 FROM ProductStocks WHERE ProductID = @ProductID)
        BEGIN
            UPDATE ProductStocks
            SET QuantityAvailable = QuantityAvailable + @QtyProduced
            WHERE ProductID = @ProductID;
        END
        ELSE
        BEGIN
            INSERT INTO ProductStocks (ProductID, QuantityAvailable,
QuantityReserved)
            VALUES (@ProductID, @QtyProduced, 0);
        END

        -- B. Rejestracja w historii ProductStockRegister
        INSERT INTO ProductStockRegister (ProductID, EntryDate, Amount)
        VALUES (@ProductID, GETDATE(), @QtyProduced);

        -- Zatwierdzenie transakcji
        COMMIT TRANSACTION;

        PRINT 'Zlecenie produkcyjne nr ' + CAST(@ProductionOrderID AS VARCHAR) + '
zostało zakończone pomyślnie.';
        PRINT 'Wyprodukowano ' + CAST(@QtyProduced AS VARCHAR) + ' szt. produktu
ID ' + CAST(@ProductID AS VARCHAR) + '.';

    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        PRINT 'Błąd podczas finalizacji produkcji: ' + ERROR_MESSAGE();
    END CATCH
END;
GO
```

GenerateInvoice

Procedura generuje fakturę dla zamówienia: oblicza kwoty netto, VAT i brutto, tworzy nagłówek faktury oraz pozycje faktury i zwraca jej ID.

```
CREATE OR ALTER PROCEDURE [dbo].[GenerateInvoice]
    @OrderID        INT,                -- Numer zamówienia
    @PaymentDays    INT = 14,           -- Termin płatności (domyślnie 14 dni)
    @NewInvoiceID   INT OUTPUT          -- ID nowej faktury
```

```
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- 0. DEKLARACJA ZMIENNYCH AGREGACYJNYCH
        DECLARE @TotalNet DECIMAL(19,4);
        DECLARE @TotalVat DECIMAL(19,4);
        DECLARE @TotalGross DECIMAL(19,4);

        -- 1. WALIDACJA WSTĘPNA
        IF NOT EXISTS (SELECT 1 FROM [dbo].[Orders] WHERE OrderID = @OrderID)
            THROW 51025, 'Błąd: Zamówienie o podanym ID nie istnieje.', 1;

        IF NOT EXISTS (SELECT 1 FROM [dbo].[OrderDetails] WHERE OrderID =
@OrderID)
            THROW 51026, 'Błąd: Zamówienie jest puste.', 1;

        IF EXISTS (SELECT 1 FROM [dbo].[Invoices] WHERE OrderID = @OrderID)
            THROW 51027, 'Błąd: Do tego zamówienia wystawiono już fakturę.', 1;

        -- 2. OBLICZENIA WARTOŚCI (Z Dynamicznym VAT)
        ;WITH OrderCalculations AS (
            SELECT
                OD.ProductID,
                P.ProductName,
                OD.Quantity,
                OD.UnitPrice AS UnitNetPrice,
                ISNULL(OD.Discount, 0) AS DiscountPercent,

                -- POBIERANIE VAT Z TABELI KATEGORII
                ISNULL(PC.VATPercentage, 23.00) AS VatRate
            FROM [dbo].[OrderDetails] OD
            JOIN [dbo].[Products] P ON OD.ProductID = P.ProductID
            -- Dołączamy kategorie, aby pobrać stawkę VAT
            LEFT JOIN [dbo].[ProductCategories] PC ON P.CategoryID = PC.CategoryID
            WHERE OD.OrderID = @OrderID
        ),
        SummaryCalculations AS (
            SELECT
                ProductID,
                ProductName,
                Quantity,
                UnitNetPrice,
                DiscountPercent,
                VatRate,
                -- Wartość Netto (po rabacie)
                (Quantity * UnitNetPrice * (1.0 - DiscountPercent/100.0)) AS
LineNetTotal,

                -- Wartość VAT: Netto * (Stawka / 100)
                ((Quantity * UnitNetPric* (1.0 - DiscountPercent/100.0)) *
```

```
(VatRate / 100.0)) AS LineVatTotal
            FROM OrderCalculations
        )
        -- 3. PRZYPISANIE DO ZMIENNYCH
        SELECT
            @TotalNet = SUM(LineNetTotal),
            @TotalVat = SUM(LineVatTotal),
            @TotalGross = SUM(LineNetTotal + LineVatTotal)
        FROM SummaryCalculations;

        -- 4. GENEROWANIE NUMERU FAKTURY
        DECLARE @InvoiceNumber VARCHAR(50);
        SET @InvoiceNumber = 'FV/' + FORMAT(GETDATE(), 'yyyy/MM') + '/' +
CAST(@OrderID AS VARCHAR(10));

        IF EXISTS (SELECT 1 FROM [dbo].[Invoices] WHERE InvoiceNumber =
@InvoiceNumber)
            THROW 51028, 'Błąd: Wygenerowany numer faktury już istnieje.', 1;

        -- 5. WSTAWIENIE NAGŁÓWKA FAKTURY
        INSERT INTO [dbo].[Invoices] (
            [OrderID],
            [InvoiceNumber],
            [InvoiceDate],
            [DueDate],
            [TotalNet],
            [TotalVat],
            [TotalGross]
        )
        VALUES (
            @OrderID,
            @InvoiceNumber,
            GETDATE(),
            DATEADD(DAY, @PaymentDays, GETDATE()),
            @TotalNet,
            @TotalVat,
            @TotalGross
        );

        SET @NewInvoiceID = SCOPE_IDENTITY();

        -- 6. WSTAWIENIE POZYCJI FAKTURY (InvoiceItems)
        INSERT INTO [dbo].[InvoiceItems] (
            [InvoiceID],
            [ProductID],
            [Description],
            [Quantity],
            [UnitNetPrice],
            [VatRate],
            [DiscountPercent]
        )
        SELECT
            @NewInvoiceID,
            P.ProductID,
```

```
            P.ProductName,
            OD.Quantity,
            OD.UnitPrice,
            ISNULL(PC.VAT, 23.00),
            ISNULL(OD.Discount, 0)
        FROM [dbo].[OrderDetails] OD
        JOIN [dbo].[Products] P ON OD.ProductID = P.ProductID
        LEFT JOIN [dbo].[ProductCategories] PC ON P.CategoryID = PC.CategoryID
        WHERE OD.OrderID = @OrderID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
GO
```

NewComplaint

Procedura rejestruje nową reklamację dla produktu w zamówieniu, przypisuje ją pracownikowi i zwraca ID reklamacji.

```
CREATE OR ALTER PROCEDURE [dbo].[sp_ZglosReklamacje]
    @OrderID        INT,            -- Numer reklamowanego zamówienia
    @ProductID      INT,            -- Reklamowany produkt
    @EmployeeID     INT,            -- Przypisany pracownik
    @Status         VARCHAR(30) = 'Zgłoszone', -- Domyślny status
    @NewComplaintID INT OUTPUT      -- Zwracane ID reklamacji
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- 1. WALIDACJA DANYCH

        IF NOT EXISTS (SELECT 1 FROM [dbo].[Orders] WHERE OrderID = @OrderID)
            THROW 51032, 'Błąd: Zamówienie o podanym ID nie istnieje.', 1;

        -- Sprawdzenie Czy produkt istnieje w reklamowanym zamówieniu
        IF NOT EXISTS (SELECT 1 FROM [dbo].[OrderDetails] WHERE OrderID = @OrderID
AND ProductID = @ProductID)
            THROW 51033, 'Błąd: Podany produkt nie znajduje się w tym
zamówieniu.', 1;

        -- Sprawdzenie czy pracownik istnieje
        IF NOT EXISTS (SELECT 1 FROM [dbo].[Employees] WHERE EmployeeID =
```

```
@EmployeeID)
            THROW 51034, 'Błąd: Pracownik o podanym ID nie istnieje.', 1;

        -- 2. REJESTRACJA REKLAMACJI
        INSERT INTO [dbo].[Complaints] (
            [OrderID],
            [ProductID],
            [EmployeeID],
            [ComplaintDate],
            [Status]
        )
        VALUES (
            @OrderID,
            @ProductID,
            @EmployeeID,
            GETDATE(),
            @Status
        );

        SET @NewComplaintID = SCOPE_IDENTITY();

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
GO
```

NewComponent

Procedura dodaje nowy komponent do systemu, inicjalizuje jego stan magazynowy i zwraca jego ID.

```
CREATE OR ALTER PROCEDURE [dbo].[NewComponent]
    @ComponentName      VARCHAR(100),
    @CategoryID         INT,
    @UnitOfMeasure      VARCHAR(20) = 'szt', -- np. szt, kg, m, m2
    @UnitCost           DECIMAL(19,4),       -- Koszt zakupu jednej jednostki
    @NewComponentID     INT OUTPUT
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRANSACTION;

        -- 1. Walidacja
        IF NOT EXISTS (SELECT 1 FROM [dbo].[ComponentCategories] WHERE
ComponentCategoryID = @CategoryID)
            THROW 51047, 'Błąd: Podana kategoria komponentu nie istnieje.', 1;
```

```
        IF @UnitCost <= 0
            THROW 51048, 'Błąd: Koszt jednostkowy (UnitCost) musi być większy od
zera.', 1;

        -- 2. Dodanie Komponentu
        INSERT INTO [dbo].[Components] (
            [ComponentName],
            [ComponentCategoryID],
            [UnitOfMeasure],
            [UnitCost]
        )
        VALUES (
            @ComponentName,
            @CategoryID,
            @UnitOfMeasure,
            @UnitCost
        );

        SET @NewComponentID = SCOPE_IDENTITY();

        -- 3. Inicjalizacja Magazynu (ComponentStocks)
        INSERT INTO [dbo].[ComponentStocks] (
            [ComponentID],
            [QuantityAvailable],
            [QuantityReserved]
        )
        VALUES (
            @NewComponentID,
            0.0000,
            0.0000
        );

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
GO
```

NewComponentCategories

Procedura tworzy nową kategorię komponentu w tabeli ComponentCategories i zwraca jej ID.

```
CREATE OR ALTER PROCEDURE [dbo].[NewComponentCategory]
    @CategoryName   VARCHAR(50),
    @NewCategoryID  INT OUTPUT
AS
BEGIN
```

```
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRANSACTION;

        IF @CategoryName IS NULL OR LTRIM(RTRIM(@CategoryName)) = ''
            THROW 51045, 'Błąd: Nazwa kategorii komponentu jest wymagana.', 1;

        IF EXISTS (SELECT 1 FROM [dbo].[ComponentCategories] WHERE CategoryName =
@CategoryName)
            THROW 51046, 'Błąd: Taka kategoria komponentu już istnieje.', 1;

        INSERT INTO [dbo].[ComponentCategories] (CategoryName)
        VALUES (@CategoryName);

        SET @NewCategoryID = SCOPE_IDENTITY();

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
GO
```

NewOrder

Procedura tworzy nowe zamówienie w tabeli Orders, ustawia domyślny status „Nowe”, uzupełnia dane dostawy i zwraca ID zamówienia.

```
CREATE OR ALTER PROCEDURE [dbo].[NewOrder]
    @CustomerID      INT,                -- Customer (Wymagane)
    @EmployeeID      INT,                -- Employee (Wymagane)
    @RequiredDate    DATETIME = NULL,    -- Przewidywana data realizacji
    @ShipperID       INT = NULL,         -- Przewoźnik (NULL, jeśli wybierany
później)
    @Freight         DECIMAL(19,4) = 0,  -- Koszt dostawy
    @ShipName        VARCHAR(100) = NULL, -- Nazwa odbiorcy (opcjonalne)
    @ShipAddress     VARCHAR(100) = NULL, -- Adres dostawy (opcjonalne)
    @ShipCity        VARCHAR(50) = NULL,
    @ShipRegion      VARCHAR(50) = NULL,
    @ShipPostalCode  VARCHAR(20) = NULL,
    @ShipCountry     VARCHAR(50) = NULL,
    @DiscountPercent DECIMAL(5,2) = 0,   -- Rabat na całe zamówienie
    @NewOrderID      INT OUTPUT          -- Zwraca ID utworzonego zamówienia
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;
```

```
        -- 1. Walidacja istnienia Klienta i Pracownika
        IF NOT EXISTS (SELECT 1 FROM [dbo].[Customers] WHERE CustomerID =
@CustomerID)
        BEGIN
            THROW 51009, 'Błąd: Podany klient (CustomerID) nie istnieje.', 1;
        END

        IF NOT EXISTS (SELECT 1 FROM [dbo].[Employees] WHERE EmployeeID =
@EmployeeID)
        BEGIN
            THROW 51010, 'Błąd: Podany pracownik (EmployeeID) nie istnieje.', 1;
        END

        -- 2. Automatyczne ustalenie ID dla statusu 'Nowe'
        DECLARE @StatusNoweID INT;
        SELECT @StatusNoweID = StatusID FROM [dbo].[OrderStatus] WHERE
StatusCategory = 'Nowe'; -- początkowo status zamówienia to nowe

        IF @StatusNoweID IS NULL
        BEGIN
            THROW 51011, 'Błąd: W tabeli OrderStatus brakuje statusu o nazwie
''Nowe''. Skonfiguruj słownik statusów.', 1;
        END

        -- 3. Logika "Inteligentnego Adresu"
        IF @ShipAddress IS NULL
        BEGIN
            SELECT
                @ShipName = CompanyName,
                @ShipAddress = Address,
                @ShipCity = City,
                @ShipRegion = Region,
                @ShipPostalCode = PostalCode,
                @ShipCountry = Country
            FROM [dbo].[Customers]
            WHERE CustomerID = @CustomerID;

            -- Dla klienta indywidualnego
            IF @ShipName IS NULL
            BEGIN
                 SELECT @ShipName = ContactName FROM [dbo].[Customers] WHERE
CustomerID = @CustomerID;
            END
        END

        -- 4. Wstawienie nagłówka zamówienia
        INSERT INTO [dbo].[Orders] (
            [CustomerID],
            [EmployeeID],
            [OrderDate],        -- Automatycznie GETDATE()
            [RequiredDate],
            [ShippedDate],      -- NULL przy utworzeniu
            [ShipperID],
```

```
            [Freight],
            [ShipName],
            [ShipAddress],
            [ShipCity],
            [ShipRegion],
            [ShipPostalCode],
            [ShipCountry],
            [OrderDiscountPercent],
            [StatusID]
        )
        VALUES (
            @CustomerID,
            @EmployeeID,
            GETDATE(),
            @RequiredDate,
            NULL,
            @ShipperID,
            @Freight,
            @ShipName,
            @ShipAddress,
            @ShipCity,
            @ShipRegion,
            @ShipPostalCode,
            @ShipCountry,
            @DiscountPercent,
            @StatusNoweID
        );

        -- 5. Pobranie ID nowego zamówienia
        SET @NewOrderID = SCOPE_IDENTITY();

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
GO
```

NewOrderStatusCategory

Procedura dodaje nowy status zamówienia do tabeli OrderStatus i zwraca jego ID.

```
CREATE OR ALTER PROCEDURE [dbo].[NewOrderStatusCategory]
    @StatusCategory VARCHAR(30),       -- Nazwa statusu
    @NewStatusID    INT OUTPUT         -- Zwracane ID nowego statusu
AS
BEGIN
    SET NOCOUNT ON;
```

```
    BEGIN TRY
        BEGIN TRANSACTION;

        -- 1. Walidacja danych wejściowych
        IF @StatusCategory IS NULL OR LTRIM(RTRIM(@StatusCategory)) = ''
        BEGIN
            THROW 51007, 'Błąd: Nazwa statusu (StatusCategory) nie może być
pusta.', 1;
        END

        -- 2. Sprawdzenie duplikatów
        IF EXISTS (SELECT 1 FROM [dbo].[OrderStatus] WHERE StatusCategory =
@StatusCategory)
        BEGIN
            THROW 51008, 'Błąd: Taki status zamówienia już istnieje w bazie
danych.', 1;
        END

        -- 3. Wstawienie rekordu
        INSERT INTO [dbo].[OrderStatus] (
            [StatusCategory]
        )
        VALUES (
            @StatusCategory
        );

        -- 4. Pobranie ID nowo dodanego statusu
        SET @NewStatusID = SCOPE_IDENTITY();

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH
END;
GO
```

NewPayment

Procedura rejestruje płatność za fakturę, uwzględnia pozostałą kwotę do zapłaty, sprawdza nadpłatę i zwraca ID nowej płatności.

```
CREATE OR ALTER PROCEDURE [dbo].[sp_RejestrujPlatnosc]
    @InvoiceID      INT,                -- Opłacana faktura
    @Amount         DECIMAL(19,4) = NULL,-- Kwota wpłaty (jeśli NULL -> wpłacana
jest całość pozostałej kwoty)
    @PaymentMethod  VARCHAR(30),        -- 'Przelew', 'Karta', 'BLIK'
```

```
    @PaymentDate    DATETIME = NULL,      -- Data wpłaty (domyślnie TERAZ)
    @NewPaymentID   INT OUTPUT            -- Zwracane ID płatności
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- 1. WALIDACJA WSTĘPNA

        -- Sprawdzenie czy faktura istnieje
        IF NOT EXISTS (SELECT 1 FROM [dbo].[Invoices] WHERE InvoiceID =
@InvoiceID)
            THROW 51035, 'Błąd: Faktura o podanym ID nie istnieje.', 1;

        -- Pobranie danych o fakturze (łączna kwota płatności)
        DECLARE @TotalGross DECIMAL(19,4);
        SELECT @TotalGross = TotalGross FROM [dbo].[Invoices] WHERE InvoiceID =
@InvoiceID;

        -- Obliczenie ile już wpłacono do tej pory
        DECLARE @AlreadyPaid DECIMAL(19,4);
        SELECT @AlreadyPaid = ISNULL(SUM(Amount), 0)
        FROM [dbo].[Payments]
        WHERE InvoiceID = @InvoiceID AND PaymentStatus = 'Zaksięgowana';

        -- Obliczenie ile zostało do zapłaty
        DECLARE @RemainingToPay DECIMAL(19,4);
        SET @RemainingToPay = @TotalGross - @AlreadyPaid;

        -- Jeśli faktura jest już opłacona w całości
        IF @RemainingToPay <= 0
            THROW 51036, 'Błąd: Ta faktura jest już w całości opłacona.', 1;


        -- 2. OBSŁUGA KWOTY WPŁATY

        -- Jeśli użytkownik nie podał kwoty (@Amount IS NULL), zakładamy spłatę
całości długu
        IF @Amount IS NULL
        BEGIN
            SET @Amount = @RemainingToPay;
        END

        -- Walidacja kwoty wpłaty
        IF @Amount <= 0
            THROW 51037, 'Błąd: Kwota płatności musi być większa od zera.', 1;

        -- Zabezpieczenie przed nadpłatą (Overpayment check)
        IF @Amount > @RemainingToPay
        BEGIN
            DECLARE @ErrMsg NVARCHAR(200) = 'Błąd: Próba nadpłaty. Pozostało do
zapłaty: ' + CAST(@RemainingToPay AS NVARCHAR(20));
```

```
            THROW 51038, @ErrMsg, 1;
        END

        -- Domyślna data
        IF @PaymentDate IS NULL SET @PaymentDate = GETDATE();


        -- 3. REJESTRACJA PŁATNOŚCI

        INSERT INTO [dbo].[Payments] (
            [InvoiceID],
            [PaymentDate],
            [Amount],
            [PaymentMethod],
            [PaymentStatus]
        )
        VALUES (
            @InvoiceID,
            @PaymentDate,
            @Amount,
            @PaymentMethod,
            'Zaksięgowana'
        );

        SET @NewPaymentID = SCOPE_IDENTITY();

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
GO
```

NewProduct

Procedura dodaje nowy produkt do tabeli Products, inicjalizuje jego stan magazynowy i zwraca jego ID.

```
CREATE OR ALTER PROCEDURE [dbo].[sp_DodajProdukt]
    @ProductName            VARCHAR(100),       -- Nazwa produktu
    @CategoryID             INT,                -- Kategoria
    @UnitPriceNet           DECIMAL(19,4),      -- Cena sprzedaży netto
    @ProductionTimeInHours  INT,                -- Czas produkcji (w godzinach)
    @ProductionCost         DECIMAL(19,4),      -- Koszt produkcji
    @NewProductID           INT OUTPUT          -- Zwracane ID nowego produktu
AS
BEGIN
    SET NOCOUNT ON;
```

```
    BEGIN TRY
        BEGIN TRANSACTION;

        -- 1. WALIDACJA DANYCH WEJŚCIOWYCH

        IF @ProductName IS NULL OR LTRIM(RTRIM(@ProductName)) = ''
        BEGIN
            THROW 51039, 'Błąd: Nazwa produktu jest wymagana.', 1;
        END

        -- Kategoria musi istnieć
        IF NOT EXISTS (SELECT 1 FROM [dbo].[ProductCategories] WHERE CategoryID =
@CategoryID)
        BEGIN
            THROW 51040, 'Błąd: Podana kategoria produktu nie istnieje.', 1;
        END

        -- 2. WALIDACJA WARTOŚCI LICZBOWYCH (Constraints CHECK)

        IF @UnitPriceNet <= 0
            THROW 51041, 'Błąd: Cena sprzedaży (UnitPriceNet) musi być większa od
zera.', 1;

        IF @ProductionTimeInHours <= 0
            THROW 51042, 'Błąd: Czas produkcji (ProductionTimeInHours) musi być
większy od zera.', 1;

        IF @ProductionCost <= 0
            THROW 51043, 'Błąd: Koszt produkcji (ProductionCost) musi być większy
od zera.', 1;

        -- 3. SPRAWDZENIE DUPLIKATÓW
        IF EXISTS (SELECT 1 FROM [dbo].[Products] WHERE ProductName =
@ProductName)
        BEGIN
            THROW 51044, 'Błąd: Produkt o podanej nazwie już istnieje.', 1;
        END

        -- 4. WSTAWIENIE PRODUKTU (Table Products)
        INSERT INTO [dbo].[Products] (
            [ProductName],
            [CategoryID],
            [UnitPriceNet],
            [ProductionTimeInHours],
            [ProductionCost]
        )
        VALUES (
            @ProductName,
            @CategoryID,
            @UnitPriceNet,
            @ProductionTimeInHours,
            @ProductionCost
        );
```

```
        SET @NewProductID = SCOPE_IDENTITY();

        -- 5. INICJALIZACJA STANU MAGAZYNOWEGO (Table ProductStocks)

        INSERT INTO [dbo].[ProductStocks] (
            [ProductID],
            [QuantityAvailable],
            [QuantityReserved]
        )
        VALUES (
            @NewProductID,
            0.00,
            0.00
        );

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
GO
```

NewWorkCentre

Procedura tworzy nowe stanowisko produkcyjne w tabeli WorkCenters, ustawia jego dzienne moce przerobowe i zwraca jego ID.

```
CREATE OR ALTER PROCEDURE [dbo].[NewWorkCentre]
    @WorkCenterName     VARCHAR(80),    -- Nazwa stanowiska Work Centre
    @CapacityHours      FLOAT,          -- Ile godzin dziennie pracuje
    @LabourRatePerHour  MONEY,          -- Koszt roboczogodziny
    @NewWorkCenterID    INT OUTPUT
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRANSACTION;

        -- 1. WALIDACJA
        IF @WorkCenterName IS NULL OR LTRIM(RTRIM(@WorkCenterName)) = ''
            THROW 51053, 'Błąd: Nazwa stanowiska jest wymagana.', 1;

        IF @CapacityHours <= 0 OR @CapacityHours > 24
            THROW 51054, 'Błąd: Moce przerobowe muszą być z zakresu 0-24 godzin.',
1;

        IF EXISTS (SELECT 1 FROM [dbo].[WorkCenters] WHERE WorkCenterName =
@WorkCenterName)
```

```
            THROW 51055, 'Błąd: Stanowisko o podanej nazwie już istnieje.', 1;

        -- 2. DODANIE STANOWISKA
        INSERT INTO [dbo].[WorkCenters] (
            [WorkCenterName],
            [IsActive]
        )
        VALUES (
            @WorkCenterName,
            1
        );

        SET @NewWorkCenterID = SCOPE_IDENTITY();

        -- 3. DEFINICJA MOCY PRZEROBOWYCH (WorkCenterCapacity)
        INSERT INTO [dbo].[WorkCenterCapacity] (
            [WorkCenterID],
            [CapacityHoursPerDay]
        )
        VALUES (
            @NewWorkCenterID,
            @CapacityHours
        );

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
GO
```

PlanProductionForOrder

Procedura analizuje stan magazynowy zamówienia i dla brakujących produktów tworzy zlecenia produkcyjne oraz rezerwuje potrzebne komponenty.

```
/*
 Opis działania procedury
 1. Analiza braków: Procedura sprawdza każdą pozycję zamówienia (OrderDetails).
    Porównuje zamówioną ilość z ilością dostępną w magazynie (ProductStocks).
 2. Tworzenie zlecenia: Jeśli brakuje towaru (ilość dostępna - ilość zarezerwowana
< ilość zamówiona),
    system tworzy nowe zlecenie produkcyjne (ProductionOrders) na brakującą liczbę
sztuk.
 3. Czas produkcji: Data zakończenia (PlannedEndDate) jest wyliczana na podstawie
czasu produkcji jednego produktu
    (ProductionTimeInHours) pomnożonego przez liczbę sztuk
 4. Rezerwacja składników: Na podstawie przepisu (BillOfMaterials) procedura
oblicza zapotrzebowanie na surowce,
```

```
    dodaje wpisy do ProductionOrderComponents oraz rezerwuje je w magazynie
(ComponentStocks i ComponentReservations).
 */

CREATE OR ALTER PROCEDURE [dbo].[PlanProductionForOrder]
    @OrderID int
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @OrderDetailID int;
    DECLARE @ProductID int;
    DECLARE @OrderQty decimal(12,2);
    DECLARE @StockAvailable decimal(12,2);
    DECLARE @StockReserved decimal(12,2);
    DECLARE @FreeStock decimal(12,2);
    DECLARE @MissingQty decimal(12,2);

    DECLARE @ProductionOrderID int;
    DECLARE @ProductionTime int;
    DECLARE @WorkCenterID int;
    DECLARE @NewStatusID int;
    DECLARE @PlannedStart datetime;
    DECLARE @PlannedEnd datetime;

    -- Pobranie ID statusu dla 'Planowane' (lub 'Nowe')
    SELECT TOP 1 @NewStatusID = ProductionOrderStatusID
    FROM ProductionOrderStatus
    WHERE StatusName LIKE 'Planowane%' OR StatusName LIKE 'Nowe%'
    ORDER BY ProductionOrderStatusID;

    IF @NewStatusID IS NULL SET @NewStatusID = 1;

    SELECT TOP 1 @WorkCenterID = WorkCenterID
    FROM WorkCenters
    WHERE IsActive = 1;

    IF @WorkCenterID IS NULL
    BEGIN
        PRINT 'Błąd: Brak aktywnych gniazd produkcyjnych (WorkCenters).';
        RETURN;
    END

    DECLARE cur_orders CURSOR FOR
    SELECT OrderDetailID, ProductID, Quantity
    FROM OrderDetails
    WHERE OrderID = @OrderID;

    OPEN cur_orders;

    FETCH NEXT FROM cur_orders INTO @OrderDetailID, @ProductID, @OrderQty;

    WHILE @@FETCH_STATUS = 0
    BEGIN
```

```
        -- Sprawdzenie stanu magazynowego produktu
        SELECT
            @StockAvailable = ISNULL(QuantityAvailable, 0),
            @StockReserved = ISNULL(QuantityReserved, 0)
        FROM ProductStocks
        WHERE ProductID = @ProductID;

        -- Jeśli nie ma wpisu w ProductStocks, przyjmujemy 0
        SET @StockAvailable = ISNULL(@StockAvailable, 0);
        SET @StockReserved = ISNULL(@StockReserved, 0);

        -- Obliczenie wolnego zapasu (Dostępne - Zarezerwowane)
        SET @FreeStock = @StockAvailable - @StockReserved;

        -- Jeśli brakuje towaru na pokrycie zamówienia
        IF @FreeStock < @OrderQty
        BEGIN
            SET @MissingQty = @OrderQty - CASE WHEN @FreeStock > 0 THEN @FreeStock
ELSE 0 END;

            -- Pobranie czasu produkcji dla produktu
            SELECT @ProductionTime = ProductionTimeInHours
            FROM Products
            WHERE ProductID = @ProductID;

            SET @PlannedStart = GETDATE();
            -- Wyliczenie daty końca: Start + (Czas jednostkowy * Ilość)
            SET @PlannedEnd = DATEADD(HOUR, ISNULL(@ProductionTime, 1) *
@MissingQty, @PlannedStart);

            -- 1. Utworzenie zlecenia produkcyjnego (ProductionOrders)
            INSERT INTO ProductionOrders (
                ProductID,
                QuantityToProduce,
                PlannedStartDate,
                PlannedEndDate,
                SourceOrderID,
                WorkCenterID,
                ProductionOrderStatusID
            )
            VALUES (
                @ProductID,
                @MissingQty,
                @PlannedStart,
                @PlannedEnd,
                @OrderID,
                @WorkCenterID,
                @NewStatusID
            );

            -- Pobranie ID nowo utworzonego zlecenia
            SET @ProductionOrderID = SCOPE_IDENTITY();

            PRINT 'Utworzono zlecenie produkcyjne ID: ' + CAST(@ProductionOrderID
```

```
AS VARCHAR) + ' dla produktu ID: ' + CAST(@ProductID AS VARCHAR);

            -- 2. Przypisanie komponentów do zlecenia (na podstawie
BillOfMaterials)
            INSERT INTO ProductionOrderComponents (
                ProductionOrderID,
                ComponentID,
                QuantityRequired,
                QuantityIssued
            )
            SELECT
                @ProductionOrderID,
                ComponentID,
                QuantityPerProduct * @MissingQty,
                0
            FROM BillOfMaterials
            WHERE ProductID = @ProductID;

            -- 3. Rezerwacja komponentów w magazynie (ComponentStocks +
ComponentReservations)

            -- Pętla po komponentach, aktualizująca rezerwacje
            DECLARE @CompID int;
            DECLARE @CompQtyNeeded decimal(12,4);

            DECLARE cur_components CURSOR FOR
            SELECT ComponentID, QuantityRequired
            FROM ProductionOrderComponents
            WHERE ProductionOrderID = @ProductionOrderID;

            OPEN cur_components;
            FETCH NEXT FROM cur_components INTO @CompID, @CompQtyNeeded;

            WHILE @@FETCH_STATUS = 0
            BEGIN
                -- A. Dodanie wpisu do tabeli rezerwacji
                INSERT INTO ComponentReservations (
                    ComponentReservationID,
                    ComponentID,
                    OrderDetailID,
                    ProductionOrderID,
                    ReservedQuantity
                )
                VALUES (
                    (SELECT ISNULL(MAX(ComponentReservationID), 0) + 1 FROM
ComponentReservations),
                    @CompID,
                    NULL,
                    @ProductionOrderID,
                    @CompQtyNeeded
                );

                -- B. Aktualizacja stanu zarezerwowanego w ComponentStocks
                IF EXISTS (SELECT 1 FROM ComponentStocks WHERE ComponentID =
```

```
@CompID)
                BEGIN
                    UPDATE ComponentStocks
                    SET QuantityReserved = ISNULL(QuantityReserved, 0) +
@CompQtyNeeded
                    WHERE ComponentID = @CompID;
                END
                ELSE
                BEGIN
                    INSERT INTO ComponentStocks (ComponentID, QuantityAvailable,
QuantityReserved)
                    VALUES (@CompID, 0, @CompQtyNeeded);
                END

                FETCH NEXT FROM cur_components INTO @CompID, @CompQtyNeeded;
            END

            CLOSE cur_components;
            DEALLOCATE cur_components;

        END
        ELSE
        BEGIN
            PRINT 'Produkt ID ' + CAST(@ProductID AS VARCHAR) + ' dostępny w
magazynie. Nie zlecono produkcji.';
        END

        FETCH NEXT FROM cur_orders INTO @OrderDetailID, @ProductID, @OrderQty;
    END

    CLOSE cur_orders;
    DEALLOCATE cur_orders;
END;
GO
```

RegisterComponentDelivery

Rejestruje dostawę komponentu, aktualizując stan magazynu i historię zmian.

```
/*
 Rejestracja dostawy: Tworzy nowy wpis w tabeli ComponentDeliveries ze statusem
"Dostarczono" (lub "Zrealizowano").

Aktualizacja stanu magazynowego: Zwiększa ilość dostępną (QuantityAvailable) w
tabeli ComponentStocks.
 Jeśli dany komponent nie widnieje jeszcze w rejestrze stanów, tworzy nowy rekord.

Historia zmian: Dodaje wpis do rejestru ComponentStockRegister, dokumentując
przyrost ilości surowca.
 */
```

```
CREATE OR ALTER PROCEDURE [dbo].[RegisterComponentDelivery]
    @ComponentID int,
    @Quantity decimal(12,4),
    @DeliveryDate date = NULL -- Opcjonalnie, domyślnie dzisiejsza data
AS
BEGIN
    SET NOCOUNT ON;

    -- Ustawienie domyślnej daty, jeśli nie podano
    IF @DeliveryDate IS NULL SET @DeliveryDate = CAST(GETDATE() AS date);

    -- Walidacja danych wejściowych
    IF @Quantity <= 0
    BEGIN
        PRINT 'Błąd: Ilość dostarczonych komponentów musi być większa od zera.';
        RETURN;
    END

    -- Sprawdzenie czy komponent istnieje
    IF NOT EXISTS (SELECT 1 FROM Components WHERE ComponentID = @ComponentID)
    BEGIN
        PRINT 'Błąd: Nie znaleziono komponentu o ID: ' + CAST(@ComponentID AS
VARCHAR);
        RETURN;
    END

    BEGIN TRANSACTION;

    BEGIN TRY
        -- 1. Dodanie wpisu do tabeli dostaw (ComponentDeliveries)
        INSERT INTO ComponentDeliveries (
            ComponentID,
            Quantity,
            DateOfDelivery,
            Status
        )
        VALUES (
            @ComponentID,
            @Quantity,
            @DeliveryDate,
            'Dostarczono'
        );

        -- Pobranie ID nowej dostawy
        DECLARE @NewDeliveryID int = SCOPE_IDENTITY();

        -- 2. Aktualizacja stanu magazynowego (ComponentStocks)
        IF EXISTS (SELECT 1 FROM ComponentStocks WHERE ComponentID = @ComponentID)
        BEGIN
            UPDATE ComponentStocks
            SET QuantityAvailable = QuantityAvailable + @Quantity
            WHERE ComponentID = @ComponentID;
        END
        ELSE
```

```
        BEGIN
            INSERT INTO ComponentStocks (ComponentID, QuantityAvailable,
QuantityReserved)
            VALUES (@ComponentID, @Quantity, 0);
        END

        -- 3. Rejestracja w historii zmian magazynowych (ComponentStockRegister)
        INSERT INTO ComponentStockRegister (
            ComponentID,
            EntryDate,
            Amount
        )
        VALUES (
            @ComponentID,
            GETDATE(),
            @Quantity
        );

        COMMIT TRANSACTION;

        PRINT 'Zarejestrowano dostawę nr ' + CAST(@NewDeliveryID AS VARCHAR) +
              '. Przyjęto ' + CAST(@Quantity AS VARCHAR) +
              ' szt. komponentu ID ' + CAST(@ComponentID AS VARCHAR) + '.';

    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        PRINT 'Błąd podczas rejestracji dostawy: ' + ERROR_MESSAGE();
    END CATCH
END;
GO
```

RegisterCustomer

Rejestruje nowego klienta w bazie, walidując dane i przypisując ID.

```
CREATE OR ALTER PROCEDURE [dbo].[RegisterCustomer]
    @CompanyName    VARCHAR(100) = NULL,
    @ContactName    VARCHAR(60) = NULL,
    @ContactTitle   VARCHAR(60) = NULL,
    @IsBusiness     BIT,
    @NIP            VARCHAR(15) = NULL,
    @Address        VARCHAR(100) = NULL,
    @City           VARCHAR(50) = NULL,
    @Region         VARCHAR(50) = NULL,
    @PostalCode     VARCHAR(20) = NULL,
    @Country        VARCHAR(50) = NULL,
    @Phone          VARCHAR(30) = NULL,
    @Email          VARCHAR(100) = NULL,
    @NewCustomerID  INT OUTPUT
AS
```

```
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- 1. Walidacja danych wejściowych

        IF @IsBusiness = 1
        BEGIN
            IF @CompanyName IS NULL OR LTRIM(RTRIM(@CompanyName)) = ''
            BEGIN
                THROW 51000, 'Błąd: Dla klienta biznesowego wymagane jest podanie
nazwy firmy (CompanyName).', 1;
            END

            IF @NIP IS NULL OR LTRIM(RTRIM(@NIP)) = ''
            BEGIN
                THROW 51001, 'Błąd: Dla klienta biznesowego wymagane jest podanie
numeru NIP.', 1;
            END
        END

        IF @IsBusiness = 0
        BEGIN
            -- Wymagane imię i nazwisko ORAZ NIP NULL
            IF @ContactName IS NULL OR LTRIM(RTRIM(@ContactName)) = ''
            BEGIN
                THROW 51002, 'Błąd: Dla klienta indywidualnego wymagane jest
podanie imienia i nazwiska (ContactName).', 1;
            END

            IF @NIP IS NOT NULL AND LTRIM(RTRIM(@NIP)) <> ''
            BEGIN
                THROW 51004, 'Błąd: Klient indywidualny nie może posiadać numeru
NIP. Pole NIP musi być puste.', 1;
            END
        END

        -- 2. Sprawdzenie unikalności NIP
        IF @NIP IS NOT NULL
        BEGIN
            IF EXISTS (SELECT 1 FROM [dbo].[Customers] WHERE NIP = @NIP)
            BEGIN
                THROW 51003, 'Błąd: Klient o podanym numerze NIP już istnieje w
bazie danych.', 1;
            END
        END

        -- 3. Wstawienie rekordu do tabeli Customers
        INSERT INTO [dbo].[Customers] (
            [CompanyName],
            [ContactName],
            [ContactTitle],
```

```
            [IsBusiness],
            [NIP],
            [Address],
            [City],
            [Region],
            [PostalCode],
            [Country],
            [Phone],
            [Email]
        )
        VALUES (
            -- Jeśli klient indywidualny, CompanyName jako NULL
            CASE WHEN @IsBusiness = 1 THEN @CompanyName ELSE NULL END,
            @ContactName,
            @ContactTitle,
            @IsBusiness,
            -- Jeśli klient indywidualny, NIP jako NULL
            CASE WHEN @IsBusiness = 1 THEN @NIP ELSE NULL END,
            @Address,
            @City,
            @Region,
            @PostalCode,
            @Country,
            @Phone,
            @Email
        );

        -- 4. Pobranie ID nowo dodanego klienta
        SET @NewCustomerID = SCOPE_IDENTITY();

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();

        RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END;
GO
```

RegisterDelivery

Rejestruje wysyłkę zamówienia, aktualizuje status zamówienia i magazyn oraz zapisuje historię wydania towaru.

```
CREATE OR ALTER PROCEDURE [dbo].[RegisterDelivery]
    @OrderID            INT,                -- ID wysyłanego zamówienia
    @ShipperID          INT,                -- Shipper ID
    @ParcelNumber       VARCHAR(30),        -- Numer listu przewozowego
    @EstimatedDays      INT = 2,            -- Szacowana liczba dni dostawy
    @NewDeliveryID      INT OUTPUT          -- Zwracane ID dostawy
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- 1. WALIDACJA DANYCH

        -- Sprawdzenie czy zamówienie istnieje
        IF NOT EXISTS (SELECT 1 FROM [dbo].[Orders] WHERE OrderID = @OrderID)
            THROW 51015, 'Błąd: Zamówienie o podanym ID nie istnieje.', 1;

        -- Sprawdzenie czy Shipper istnieje
        IF NOT EXISTS (SELECT 1 FROM [dbo].[Shippers] WHERE ShipperID =
@ShipperID)
            THROW 51016, 'Błąd: Wybrany kurier (ShipperID) nie istnieje.', 1;

        -- Sprawdzenie unikalności numeru paczki
        IF EXISTS (SELECT 1 FROM [dbo].[Deliveries] WHERE ParcelNumber =
@ParcelNumber)
            THROW 51017, 'Błąd: Ten numer przesyłki (ParcelNumber) jest już
zarejestrowany w systemie.', 1;

        -- Sprawdzenie czy zamówienie nie zostało już wysłane (ShippedDate musi
być NULL)
        IF EXISTS (SELECT 1 FROM [dbo].[Orders] WHERE OrderID = @OrderID AND
ShippedDate IS NOT NULL)
            THROW 51018, 'Błąd: To zamówienie zostało już wysłane.', 1;


        -- 2. TWORZENIE DOSTAWY

        DECLARE @ShippingDate DATETIME = GETDATE();
        DECLARE @EstimatedDate DATETIME = DATEADD(DAY, @EstimatedDays,
@ShippingDate);

        INSERT INTO [dbo].[Deliveries] (
            [ParcelNumber],
            [ShippingDate],
            [EstimatedDeliveryDate],
            [ShipperID],
            [OrderID],
            [DeliveryStatus]
        )
        VALUES (
            @ParcelNumber,
```

```
            @ShippingDate,
            @EstimatedDate,
            @ShipperID,
            @OrderID,
            'W drodze'
        );

        SET @NewDeliveryID = SCOPE_IDENTITY();


        -- 3. AKTUALIZACJA ZAMÓWIENIA (Tabela Orders)

        -- Pobranie ID statusu 'Wysłane'
        DECLARE @StatusWyslaneID INT;
        SELECT @StatusWyslaneID = StatusID FROM [dbo].[OrderStatus] WHERE
StatusCategory = 'Wysłane';

        -- Brak statusu -> błąd
        IF @StatusWyslaneID IS NULL
            THROW 51019, 'Błąd: Brak statusu ''Wysłane'' w tabeli OrderStatus.',
1;

        UPDATE [dbo].[Orders]
        SET
            ShippedDate = @ShippingDate,
            ShipperID = @ShipperID,
            StatusID = @StatusWyslaneID
        WHERE OrderID = @OrderID;


        -- 4. CYKLE MAGAZYNOWE, KONIEC REZERWACJI

        -- KROK A: Usunięcie towaru z rezerwacji (ProductStocks)
        UPDATE S
        SET QuantityReserved = S.QuantityReserved - OD.Quantity
        FROM [dbo].[ProductStocks] S
        INNER JOIN [dbo].[OrderDetails] OD ON S.ProductID = OD.ProductID
        WHERE OD.OrderID = @OrderID;

        -- KROK B: Update ProductStockRegister
        INSERT INTO [dbo].[ProductStockRegister] (
            [ProductID],
            [EntryDate],
            [Amount]
        )
        SELECT
            ProductID,
            GETDATE(),
            -Quantity
        FROM [dbo].[OrderDetails]
        WHERE OrderID = @OrderID;

        COMMIT TRANSACTION;
    END TRY
```

```
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
GO
```

RegisterShipper

Rejestruje nowego przewoźnika w systemie i zwraca jego ID.

```
CREATE OR ALTER PROCEDURE [dbo].[RegisterShipper]
    @CompanyName    VARCHAR(100),        -- Nazwa firmy przewozowej (Wymagana)
    @Phone          VARCHAR(30) = NULL,  -- Telefon kontaktowy
    @Address        VARCHAR(100) = NULL, -- Adres siedziby
    @ContactEmail   VARCHAR(100) = NULL, -- Email kontaktowy
    @NewShipperID   INT OUTPUT           -- Zwracane ID nowego przewoźnika
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- 1. Walidacja danych wejściowych

        -- Nazwa firmy nie może być pusta
        IF @CompanyName IS NULL OR LTRIM(RTRIM(@CompanyName)) = ''
        BEGIN
            THROW 51005, 'Błąd: Nazwa przewoźnika (CompanyName) jest wymagana.',
1;
        END

        -- 2. Sprawdzenie duplikatów
        IF EXISTS (SELECT 1 FROM [dbo].[Shippers] WHERE CompanyName =
@CompanyName)
        BEGIN
            THROW 51006, 'Błąd: Przewoźnik o podanej nazwie już istnieje w
systemie.', 1;
        END

        -- 3. Wstawienie rekordu
        INSERT INTO [dbo].[Shippers] (
            [CompanyName],
            [Phone],
            [Address],
            [ContactEmail]
        )
        VALUES (
            @CompanyName,
```

```
            @Phone,
            @Address,
            @ContactEmail
        );

        -- 4. Pobranie wygenerowanego ID
        SET @NewShipperID = SCOPE_IDENTITY();

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
GO
```

UpdateProductProductionCost

Aktualizuje koszt produkcji produktu lub wszystkich produktów, sumując koszt materiałów i robocizny.

```
CREATE PROCEDURE [dbo].[UpdateProductProductionCost]
    @SingleProductID int = NULL -- Opcjonalnie: ID produktu. NULL -> aktualizacja
wszystkich produktów.
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @LaborRate money;

    -- 1. Pobranie aktualnej stawki roboczogodziny
    SELECT TOP 1 @LaborRate = LabourRatePerHour
    FROM WorkingCostParameters;

    IF @LaborRate IS NULL SET @LaborRate = 0;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- Wyliczenia kosztu materiałów dla produktów
        WITH MaterialCosts AS (
            SELECT
                B.ProductID,
                SUM(B.QuantityPerProduct * C.UnitCost) AS TotalMaterialCost
            FROM BillOfMaterials B
            JOIN Components C ON B.ComponentID = C.ComponentID
            GROUP BY B.ProductID
        )
        -- Aktualizacja tabeli Products
        UPDATE P
```

```
        SET ProductionCost =
            (
                -- A. Koszt materiałów
                ISNULL(MC.TotalMaterialCost, 0)
                +
                -- B. Koszt robocizny (Czas * Stawka)
                (ISNULL(P.ProductionTimeInHours, 0) * @LaborRate)
            )
        FROM Products P
        LEFT JOIN MaterialCosts MC ON P.ProductID = MC.ProductID
        WHERE
            (@SingleProductID IS NULL OR P.ProductID = @SingleProductID);

        COMMIT TRANSACTION;

        IF @SingleProductID IS NOT NULL
            PRINT 'Zaktualizowano koszt produkcji dla produktu ID: ' +
CAST(@SingleProductID AS VARCHAR);
        ELSE
            PRINT 'Zaktualizowano koszty produkcji dla wszystkich produktów.';

    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        PRINT 'Błąd podczas aktualizacji kosztów: ' + ERROR_MESSAGE();
    END CATCH
END;
GO
```

Funkcje

Poniżej przedstawiono zestaw funkcji zaprojektowanych w bazie danych.

CalculateOrderTotalBrutto

Aktualizuje koszt produkcji produktu lub wszystkich produktów, sumując koszt materiałów i robocizny.

```
/*
 PRZYKŁAD: SELECT dbo.fn_CalculateOrderTotalBrutto(102) AS WartoscZamowienia
 */

CREATE FUNCTION fn_CalculateOrderTotalBrutto (@OrderID int)
RETURNS decimal(19,2)
AS
BEGIN
    DECLARE @TotalGross decimal(19,4);
    DECLARE @Freight decimal(19,4);
    DECLARE @GlobalDiscountPercent decimal(5,2);

    -- 1. Obliczenie sumy brutto z pozycji zamówienia
    -- Wzór: Ilość * Cena * (1 - RabatLiniowy) * (1 + StawkaVAT)
    SELECT @TotalGross = SUM(
        (OD.Quantity * OD.UnitPrice * (1 - ISNULL(OD.Discount, 0))) -- Wartość
Netto po rabacie pozycji
        * (1 + ISNULL(PC.ProductVatRate, 0)) -- Przeliczenie na Brutto wg stawki
VAT kategorii
    )
    FROM OrderDetails OD
    INNER JOIN Products P ON OD.ProductID = P.ProductID
    INNER JOIN ProductCategories PC ON P.CategoryID = PC.CategoryID
    WHERE OD.OrderID = @OrderID;

    -- brak pozycji -> ustaw 0
    SET @TotalGross = ISNULL(@TotalGross, 0);

    -- 2. Pobranie informacji o rabacie globalnym i kosztach wysyłki
    SELECT
        @Freight = Freight,
        @GlobalDiscountPercent = OrderDiscountPercent
    FROM Orders
    WHERE OrderID = @OrderID;

    -- 3. Zastosowanie rabatu globalnego dla zamówienia (jesli istnieje)
    IF @GlobalDiscountPercent > 0
    BEGIN
        SET @TotalGross = @TotalGross * (1 - (@GlobalDiscountPercent / 100.0));
    END

    -- 4. Doliczenie kosztu transportu
```

```
    SET @TotalGross = @TotalGross + ISNULL(@Freight, 0);

    -- wynik zaokrąglony do 2 miejsc po przecinku
    RETURN CAST(@TotalGross AS decimal(19,2));
END;
GO
```

EstimateProductionCompletionDate

Szacuje datę zakończenia zlecenia produkcyjnego na podstawie czasu produkcji i mocy stanowiska.

```
/*
 PRZYKŁAD:
 SELECT
    ProductionOrderID,
    PlannedStartDate,
    dbo.fn_EstimateProductionCompletionDate(ProductionOrderID) AS
ObliczonaDataKonca,
    PlannedEndDate AS PlanowanaDataKonca
FROM ProductionOrders;
 */

CREATE FUNCTION fn_EstimateProductionCompletionDate (@ProductionOrderID int)
RETURNS datetime
AS
BEGIN
    DECLARE @PlannedStart datetime;
    DECLARE @TotalHoursRequired decimal(12,2);
    DECLARE @DailyCapacityHours float;
    DECLARE @DaysRequired float;
    DECLARE @EstimatedEndDate datetime;

    -- 1. Pobranie danych o zleceniu
    SELECT
        @PlannedStart = PO.PlannedStartDate,
        -- Obliczenie łącznej liczby godzin (Ilość * Czas na sztukę)
        @TotalHoursRequired = PO.QuantityToProduce * P.ProductionTimeInHours,
        -- Pobranie wydajności dziennej (Capacity)
        @DailyCapacityHours = WCC.CapacityHoursPerDay
    FROM ProductionOrders PO
    INNER JOIN Products P ON PO.ProductID = P.ProductID
    LEFT JOIN WorkCenterCapacity WCC ON PO.WorkCenterID = WCC.WorkCenterID
    WHERE PO.ProductionOrderID = @ProductionOrderID;

    -- Zabezpieczenie: Jeśli nie znaleziono zlecenia -> NULL
    IF @PlannedStart IS NULL RETURN NULL;

    -- 2. Obliczenie daty końcowej z uwzględnieniem mocy przerobowych
    IF @DailyCapacityHours IS NOT NULL AND @DailyCapacityHours > 0
    BEGIN
        -- Dzielenie pracy na dni
```

```
        -- Przykład: 20 godzin pracy / 8 godzin dziennie = 2.5 dnia
        SET @DaysRequired = @TotalHoursRequired / @DailyCapacityHours;

        -- Dodajemy wyliczone dni do daty startu
        SET @EstimatedEndDate = DATEADD(HOUR, (@DaysRequired * 24),
@PlannedStart);
    END
    ELSE
    BEGIN
        -- Jeśli nie zdefiniowano WorkCenterCapacity czas pracy dodawany ciągiem
        SET @EstimatedEndDate = DATEADD(HOUR, @TotalHoursRequired, @PlannedStart);
    END

    RETURN @EstimatedEndDate;
END;
GO
```

GetCurrentProductionCost

Oblicza aktualny koszt produkcji produktu na podstawie kosztów materiałów i robocizny.

```
/*
 PRZYKŁAD: UPDATE Products
SET ProductionCost = dbo.fn_GetCurrentProductionCost(ProductID);
 */

CREATE FUNCTION fn_GetCurrentProductionCost (@ProductID int)
RETURNS decimal(19,4)
AS
BEGIN
    DECLARE @MaterialCost decimal(19,4);
    DECLARE @LaborCost decimal(19,4);
    DECLARE @LaborRate money;
    DECLARE @ProductionTime int;

    -- 1. Obliczenie kosztu materiałów
    -- (Suma: ilość z BillOfMaterials * aktualna cena jednostkowa komponentu)
    SELECT @MaterialCost = SUM(B.QuantityPerProduct * C.UnitCost)
    FROM BillOfMaterials B
    INNER JOIN Components C ON B.ComponentID = C.ComponentID
    WHERE B.ProductID = @ProductID;

    -- 2. Pobranie danych do kosztu robocizny
    -- Pobieramy czas produkcji przypisany do produktu
    SELECT @ProductionTime = ProductionTimeInHours
    FROM Products
    WHERE ProductID = @ProductID;

    -- Pobieramy aktualną stawkę godzinową z tabeli parametrów
    SELECT TOP 1 @LaborRate = LabourRatePerHour
    FROM WorkingCostParameters;
```

```
    -- 3. Wyliczenie kosztu robocizny (Czas * Stawka)
    SET @LaborCost = ISNULL(@ProductionTime, 0) * ISNULL(@LaborRate, 0);

    -- 4. Zwrócenie sumy całkowitej
    RETURN ISNULL(@MaterialCost, 0) + @LaborCost;
END;
GO
```

GetCustomerDiscountLevel

Wylicza poziom rabatu klienta na podstawie wartości jego zamówień z ostatniego roku.

```
CREATE FUNCTION fn_GetCustomerDiscountLevel (@CustomerID int)
RETURNS decimal(5,2)
AS
BEGIN
    DECLARE @TotalSales decimal(19,4);
    DECLARE @SuggestedDiscount decimal(5,2);

    -- 1. Obliczenie sumy wartości zamówień klienta z ostatnich 365 dni
    -- SUMA (Ilość * Cena jednostkowa) z tabeli OrderDetails
    SELECT @TotalSales = SUM(OD.Quantity * OD.UnitPrice)
    FROM Orders O
    INNER JOIN OrderDetails OD ON O.OrderID = OD.OrderID
    INNER JOIN OrderStatus OS ON O.StatusID = OS.StatusID
    WHERE O.CustomerID = @CustomerID
      AND O.OrderDate >= DATEADD(day, -365, GETDATE()) -- Tylko ostatni rok
      AND OS.StatusCategory NOT LIKE 'Anulowane%'

    --Jeśli klient nie ma historii, suma rabatu 0
    SET @TotalSales = ISNULL(@TotalSales, 0);

    -- 2. Progi rabatowe
    IF @TotalSales > 100000.00
        SET @SuggestedDiscount = 10.00; -- 10% dla klientów >100k
    ELSE IF @TotalSales > 50000.00
        SET @SuggestedDiscount = 5.00;  -- 5% dla klientów >50k
    ELSE IF @TotalSales > 10000.00
        SET @SuggestedDiscount = 2.00;  -- 2% dla klientów >10k
    ELSE
        SET @SuggestedDiscount = 0.00;  -- Brak rabatu dla pozostałych klientów

    RETURN @SuggestedDiscount;
END;
GO
```

GetMaterialShortages

Zwraca listę brakujących komponentów dla zlecenia produkcyjnego na podstawie zapotrzebowania i stanów magazynowych.

```
/*
 PRZYKŁAD
 SELECT * FROM fn_GetMaterialShortages(105);
 */
CREATE FUNCTION fn_GetMaterialShortages (@ProductionOrderID int)
RETURNS TABLE
AS
RETURN
(
    SELECT
        C.ComponentID,
        C.ComponentName,
        C.UnitOfMeasure,

        -- Łączne zapotrzebowanie dla poszczególnego zlecenia
        POC.QuantityRequired,

        -- Ile jest aktualnie na produckji
        POC.QuantityIssued,

        -- Ile jeszcze trzeba dać na produkcję (Zapotrzebowanie - Issued)
        (POC.QuantityRequired - POC.QuantityIssued) AS QuantityStillNeeded,

        -- Ile jest wolnych zasobów w magazynie (Dostępne - Zarezerwowane)
        (ISNULL(S.QuantityAvailable, 0) - ISNULL(S.QuantityReserved, 0)) AS
WarehouseFreeStock,

        -- Wyliczenie braku:
        CASE
            WHEN (POC.QuantityRequired - POC.QuantityIssued) >
(ISNULL(S.QuantityAvailable, 0) - ISNULL(S.QuantityReserved, 0))
            THEN (POC.QuantityRequired - POC.QuantityIssued) -
(ISNULL(S.QuantityAvailable, 0) - ISNULL(S.QuantityReserved, 0))
            ELSE 0
        END AS MissingQuantity

    FROM ProductionOrderComponents POC
    INNER JOIN Components C ON POC.ComponentID = C.ComponentID
    LEFT JOIN ComponentStocks S ON C.ComponentID = S.ComponentID
    WHERE POC.ProductionOrderID = @ProductionOrderID
);
GO
```

GetMaxPossibleProduction

Oblicza maksymalną liczbę sztuk produktu możliwych do wyprodukowania na podstawie dostępnych komponentów.

```
/*
 PRZYKŁAD
 SELECT
    ProductName,
    QuantityAvailable AS W_Magazynie_Gotowe,
    dbo.fn_GetMaxPossibleProduction(ProductID) AS Mozliwe_Do_Produkcji
FROM Products;
 */

CREATE FUNCTION fn_GetMaxPossibleProduction (@ProductID int)
RETURNS int
AS
BEGIN
    DECLARE @MaxPossible int;

    -- Obliczamy limit produkcji wyznaczony przez każdy składnik z osobna i
wybieramy najmniejszą wartość
    SELECT @MaxPossible = MIN(
        FLOOR(
            (ISNULL(S.QuantityAvailable, 0) - ISNULL(S.QuantityReserved, 0)) --
Ilość wolna
            /
            NULLIF(BOM.QuantityPerProduct, 0)
        )
    )
    FROM BillOfMaterials BOM
    -- LEFT JOIN sytuacja, gdy składnik jest w BillOfMaterials, ale nie ma go w
tabeli ComponentStocks (wtedy NULL -> 0)
    LEFT JOIN ComponentStocks S ON BOM.ComponentID = S.ComponentID
    WHERE BOM.ProductID = @ProductID;

    IF @MaxPossible IS NULL OR @MaxPossible < 0
        SET @MaxPossible = 0;

    RETURN @MaxPossible;
END;
GO
```

Triggery

Poniżej przedstawiono zestaw triggerów utworzonych w bazie danych. Służą one zachowaniu integralności danych w bazie, a także umożliwiają zgrabne wykonywanie operacji niemożliwych/bardzo skomplikowanych do osiągnięcia samymi zapytaniami SQL.

1. Trigger trg_OrderDetails_ReserveStock

Automatyzuje on rezerwację produktu po dodaniu pozycji zamówienia.

```
CREATE TRIGGER trg_OrderDetails_ReserveStock
ON dbo.OrderDetails
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (
        SELECT 1
        FROM inserted i
        JOIN dbo.ProductStocks ps ON ps.ProductID = i.ProductID
        GROUP BY ps.ProductID, ps.QuantityAvailable, ps.QuantityReserved
        HAVING SUM(i.Quantity) > (ps.QuantityAvailable - ps.QuantityReserved)
    )
    BEGIN
        RAISERROR ('Brak wystarczającego stanu magazynowego produktu', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END;

    UPDATE ps
    SET QuantityReserved = QuantityReserved + r.SumQty
    FROM dbo.ProductStocks ps
    JOIN (
        SELECT ProductID, SUM(Quantity) AS SumQty
        FROM inserted
        GROUP BY ProductID
    ) r ON r.ProductID = ps.ProductID;
END;
```

2. Trigger trg_OrderDetails_ReleaseStock

Umożliwia on zwolnienie rezerwacji po usunięciu pozycji zamówienia.

```
CREATE TRIGGER trg_OrderDetails_ReleaseStock
ON dbo.OrderDetails
AFTER DELETE
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE ps
    SET QuantityReserved = QuantityReserved - r.SumQty
    FROM dbo.ProductStocks ps
    JOIN (
        SELECT ProductID, SUM(Quantity) AS SumQty
        FROM deleted
        GROUP BY ProductID
    ) r ON r.ProductID = ps.ProductID;
END;
```

3. Trigger trg_OrderDetails_UpdateQuantity

Dzięki niemu możiwa jest korekta rezerwacji przy zmianie ilości zamawianego produktu.

```
CREATE TRIGGER trg_OrderDetails_UpdateQuantity
ON dbo.OrderDetails
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    IF NOT UPDATE(Quantity)
        RETURN;

    IF EXISTS (
        SELECT 1
        FROM inserted i
        JOIN deleted d ON i.OrderDetailID = d.OrderDetailID
        JOIN dbo.ProductStocks ps ON ps.ProductID = i.ProductID
        WHERE i.Quantity > d.Quantity
        GROUP BY ps.ProductID, ps.QuantityAvailable, ps.QuantityReserved
        HAVING SUM(i.Quantity - d.Quantity) >
               (ps.QuantityAvailable - ps.QuantityReserved)
    )
    BEGIN
        RAISERROR ('Brak wystarczającego stanu magazynowego przy zmianie ilości',
16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END;

    UPDATE ps
    SET QuantityReserved = QuantityReserved + x.DiffQty
    FROM dbo.ProductStocks ps
    JOIN (
        SELECT i.ProductID,
               SUM(i.Quantity - d.Quantity) AS DiffQty
        FROM inserted i
        JOIN deleted d ON i.OrderDetailID = d.OrderDetailID
        GROUP BY i.ProductID
    ) x ON x.ProductID = ps.ProductID;
END;
```

4. Trigger trg_OrderDetails_SetInitialStatus

Ustawia on początkowy status zamówienia.

```
CREATE TRIGGER trg_OrderDetails_SetInitialStatus
ON dbo.OrderDetails
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE od
    SET Status = 'Zarezerwowana'
    FROM dbo.OrderDetails od
    JOIN inserted i ON i.OrderDetailID = od.OrderDetailID;
END;
```

5. Trigger trg_InvoiceItems_RecalculateInvoice

Ten trigger przelicza wartość faktury po jakiejkolwiek zmianie w jej obrębie.

```
CREATE TRIGGER trg_InvoiceItems_RecalculateInvoice
ON dbo.InvoiceItems
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @Invoices TABLE (InvoiceID INT);

    INSERT INTO @Invoices
    SELECT DISTINCT InvoiceID FROM inserted
    UNION
    SELECT DISTINCT InvoiceID FROM deleted;

    UPDATE i
    SET
        TotalNet   = x.TotalNet,
        TotalVat   = x.TotalVat,
        TotalGross = x.TotalNet + x.TotalVat
    FROM dbo.Invoices i
    JOIN (
        SELECT
            InvoiceID,
            SUM(
                Quantity * UnitNetPrice *
                (1 - ISNULL(DiscountPercent,0)/100.0)
            ) AS TotalNet,
            SUM(
                Quantity * UnitNetPrice *
                (1 - ISNULL(DiscountPercent,0)/100.0) *
                ISNULL(VatRate, 23)/100.0
            ) AS TotalVat
        FROM dbo.InvoiceItems
        GROUP BY InvoiceID
    ) x ON x.InvoiceID = i.InvoiceID
    WHERE i.InvoiceID IN (SELECT InvoiceID FROM @Invoices);
END;
```

6. Trigger trg_Payments_PreventOverpayment

Ten trigger uniemożliwia nadpłatę faktury, w jej przypadku robi rollback transakcji.

```
CREATE TRIGGER trg_Payments_PreventOverpayment
ON dbo.Payments
AFTER INSERT, UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (
        SELECT 1
        FROM inserted p
        JOIN dbo.Invoices i ON i.InvoiceID = p.InvoiceID
        GROUP BY p.InvoiceID, i.TotalGross
        HAVING
            ISNULL(SUM(ISNULL(p.Amount,0)),0) +
            ISNULL((
                SELECT SUM(ISNULL(Amount,0))
                FROM dbo.Payments
                WHERE InvoiceID = p.InvoiceID
            ),0)
            > i.TotalGross
    )
    BEGIN
        RAISERROR ('Nadpłata faktury', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END;
END;
```

7. Trigger trg_Payments_SetStatus

Ten trigger automatycznie ustawia status płatności.

```
CREATE TRIGGER trg_Payments_SetStatus
ON dbo.Payments
AFTER INSERT, UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE p
    SET PaymentStatus =
        CASE
            WHEN ISNULL(p.Amount,0) = 0 THEN 'Nieopłacona'
            ELSE 'Zaksięgowana'
        END
    FROM dbo.Payments p
    JOIN inserted i ON i.PaymentID = p.PaymentID;
END;
```

8. Trigger trg_Invoices_CheckDates

Dzięki niemu mamy walidację dat faktury.

```
CREATE TRIGGER trg_Invoices_CheckDates
ON dbo.Invoices
AFTER INSERT, UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (
        SELECT 1
        FROM inserted
        WHERE DueDate < InvoiceDate
    )
    BEGIN
        RAISERROR ('Termin płatności nie może być wcześniejszy niż data faktury',
16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END;
END;
```

9. Trigger trg_Payments_UpdateInvoiceStatus

Ten trigger aktualizuje status całej faktury na podstawie statusu płatności.

```
CREATE TRIGGER trg_Payments_UpdateInvoiceStatus
ON dbo.Payments
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @Invoices TABLE (InvoiceID INT);

    INSERT INTO @Invoices
    SELECT DISTINCT InvoiceID FROM inserted
    UNION
    SELECT DISTINCT InvoiceID FROM deleted;

    UPDATE i
    SET Status =
        CASE
            WHEN ISNULL(p.SumPaid,0) = 0 THEN 'Nieopłacona'
            WHEN p.SumPaid < i.TotalGross THEN 'Częściowo opłacona'
            ELSE 'Opłacona'
        END
    FROM dbo.Invoices i
    LEFT JOIN (
        SELECT InvoiceID, SUM(ISNULL(Amount,0)) AS SumPaid
        FROM dbo.Payments
        GROUP BY InvoiceID
    ) p ON p.InvoiceID = i.InvoiceID
    WHERE i.InvoiceID IN (SELECT InvoiceID FROM @Invoices);
END;
```

10. Trigger trg_InvoiceItems_MarkOrderDetails

Ten trigger oznacza poszczególne części zamówień jako zafakturowane.

```
CREATE TRIGGER trg_InvoiceItems_MarkOrderDetails
ON dbo.InvoiceItems
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE od
    SET Status = 'Zafakturowana'
    FROM dbo.OrderDetails od
    JOIN dbo.Invoices i ON i.OrderID = od.OrderID
    JOIN inserted ii ON ii.InvoiceID = i.InvoiceID
                    AND ii.ProductID = od.ProductID;
END;
```

Uprawnienia

Biorąc pod uwagę specyfikę projektu, proponujemy następujące uprawnienia dostępu do danych:

1. Administrator

- zarządza systemem i danymi
- widzi wszystko (pełny dostęp do danych)

2. Pracownik biurowy (obsługa zamówień)

- przyjmuje zamówienia
- nadzoruje realizację

3. Pracownik produkcji

- widzi, co ma być wyprodukowane
- aktualizuje status produkcji

4. Przewoźnik

- widzi tylko zamówienia do dostarczenia
- aktualizuje status dostawy

Generowanie przykładowych danych

Dane do tabel, które zawierają niedużo wierszy (np. Customers, Shippers) zostały ręcznie wprowadzone do bazy. Przykładowe dane dla tych tabel wygenerowane zostały przy użyciu Gemini 3 Pro, a następnie manualnie wprowadzone do bazy.

Zamówienia i zawartości zamówień wygenerowane zostały przy użyciu kody w python.

Generowanie elementów tabeli Orders

```
import random
from datetime import date, timedelta

# Konfiguracja
FILENAME = "insert_orders.sql"
ILOSC_REKORDOW = 200

# Zakres dat: 2026-01-20 do 2026-01-30
START_DATE = date(2026, 1, 20)
END_DATE = date(2026, 1, 30)
DAYS_DIFF = (END_DATE - START_DATE).days

def generate_sql_file():
    with open(FILENAME, "w", encoding="utf-8") as f:
        # Nagłówek pliku
        f.write("USE [projekt]; -- Upewnij się, że to dobra baza\n")
        f.write("GO\n\n")
        f.write("DECLARE @MyNewOrderID INT;\n\n")

        print(f"Generowanie {ILOSC_REKORDOW} rekordów...")

        for i in range(ILOSC_REKORDOW):
            # 1. Losowanie danych
            customer_id = random.randint(35, 47)
            employee_id = random.randint(10, 18)
            shipper_id = random.randint(4, 6)

            # Losowanie daty: start + losowa liczba dni
            random_days_add = random.randint(0, DAYS_DIFF)
            required_date = START_DATE + timedelta(days=random_days_add)
            required_date_str = required_date.strftime("%Y-%m-%d")

            # 2. Budowanie tekstu zapytania
            sql_stm = (
                f"-- Rekord {i+1}\n"
                f"EXEC [dbo].[sp_NewOrder]\n"
                f"    @CustomerID = {customer_id},\n"
                f"    @EmployeeID = {employee_id},\n"
                f"    @RequiredDate = '{required_date_str}',\n"
                f"    @ShipperID = {shipper_id},\n"
```

```
                f"    @NewOrderID = @MyNewOrderID OUTPUT;\n\n"
            )

            # 3. Zapis do pliku
            f.write(sql_stm)

    print(f"Gotowe! Wynik zapisano w pliku: {FILENAME}")

if __name__ == "__main__":
    generate_sql_file()
```

Przykład

```
EXEC [dbo].[sp_NewOrder]
    @CustomerID = 36,
    @EmployeeID = 14,
    @RequiredDate = '2026-01-25',
    @ShipperID = 4,
    @NewOrderID = @MyNewOrderID OUTPUT;
```

Generowanie elementów tabeli OrderDetails

Do utworzonych zamówień dodane zostały produkty przy użyciu wygenerowanego w python kodu SQL

```
import random

# Konfiguracja zakresów
FILENAME = "insert_positions.sql"

ORDER_START = 4
ORDER_END = 204  # Włącznie

PRODUCT_START = 12
PRODUCT_END = 25 # Włącznie

QTY_MIN = 1
QTY_MAX = 100

POSITIONS_PER_ORDER_MIN = 1
POSITIONS_PER_ORDER_MAX = 10

def generate_positions_sql():
    # Przygotowanie puli wszystkich możliwych produktów
    all_products_pool = list(range(PRODUCT_START, PRODUCT_END + 1))

    total_inserts = 0

    with open(FILENAME, "w", encoding="utf-8") as f:
        f.write("USE [projekt];\n")
```

```
        f.write("GO\n\n")

        print(f"Generowanie pozycji dla zamówień {ORDER_START}-{ORDER_END}...")

        # Pętla po zamówieniach
        for order_id in range(ORDER_START, ORDER_END + 1):

            # 1. Liczba pozycji ile ma mieć to zamówienie
            num_positions = random.randint(POSITIONS_PER_ORDER_MIN,
POSITIONS_PER_ORDER_MAX)

            # 2. UNIKALNE produkty z puli dla tego zamówienia
            selected_products = random.sample(all_products_pool, k=num_positions)

            f.write(f"-- Zamówienie ID: {order_id} (Liczba pozycji:
{num_positions})\n")

            # 3. Generowanie insertów dla wylosowanych produktów
            for product_id in selected_products:
                quantity = random.randint(QTY_MIN, QTY_MAX)

                sql_stm = (
                    f"EXEC [dbo].[sp_AddPositionToOrder]\n"
                    f"    @OrderID = {order_id},\n"
                    f"    @ProductID = {product_id},\n"
                    f"    @Quantity = {quantity};\n"
                )
                f.write(sql_stm)
                total_inserts += 1

            f.write("\n")

    print(f"Gotowe! Wygenerowano {total_inserts} instrukcji INSERT w pliku:
{FILENAME}")

if __name__ == "__main__":
    generate_positions_sql()
```

Przykład

```
EXEC [dbo].[sp_AddPositionToOrder]
    @OrderID = 4,
    @ProductID = 20,
    @Quantity = 53;
```

Pozostałe generatory

Pozostałe wygenerowane elementy bazy zostały zaimplementowane do bazy i są dostępne do wyeksportowania po poprawnym połączeniem z bazą na serwerze Azure