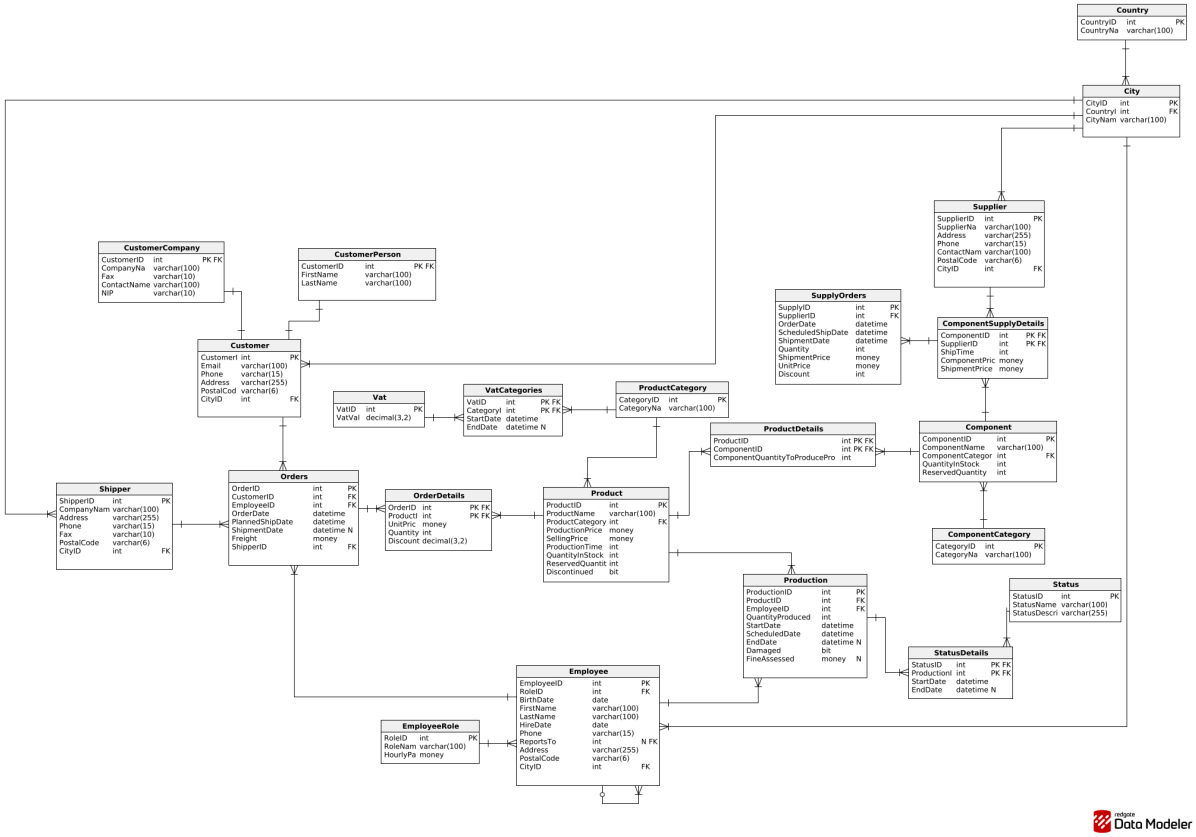
Sprawozdanie

1. Obserwacje

Baza danych powinna zawierać informacje na temat rodzajów produktów (tak jak to było w Northwind tabela Products). Również powinna zawierać informacje na temat historii sprzedaży. W bazie danych należałoby umieścić informacje jakie surowce są potrzebne do stworzenia produktu końcowego, czas oraz koszt wykonania produktu. Baza danych ma za zadanie przedstawiać stan magazynu oraz planowane elementy do produkcji. W stanie magazynu warto by zawrzeć informacje na temat planowanych produktów do produkcji. Dane uwzględniają moce przerobowe w jednostce czasu. Baza zawiera historie sprzedaży i naliczone rabaty.

2.Schemat bazy danych:



3.Tabele

3.1. City - słownikowa tabela przechowująca listę miast

- CityID: Unikalny identyfikator miasta (klucz główny).
- CountryID: Identyfikator kraju, do którego należy miasto (klucz obcy).
- CityName: Nazwa miasta.

```
SQL
CREATE TABLE City (
    CityID int  NOT NULL,
    CountryID int  NOT NULL,
    CityName varchar(100)  NOT NULL,
    CONSTRAINT City_pk PRIMARY KEY  (CityID)
);
```

3.2. Component (Komponenty/Półprodukty) - główny magazyn części i surowców używanych do produkcji mebli.

- ComponentID: Unikalny identyfikator komponentu
- ComponentName: Nazwa części
- ComponentCategoryID: Przypisanie do kategorii
- QuantityInStock: Fizyczna liczba sztuk dostępna w magazynie.
- ReservedQuantity: Liczba sztuk zarezerwowana pod zaplanowaną produkcję

```
SQL
CREATE TABLE Component (
    ComponentID int  NOT NULL,
    ComponentName varchar(100)  NOT NULL,
    ComponentCategoryID int  NOT NULL,
    QuantityInStock int  NOT NULL,
    ReservedQuantity int  NOT NULL,
    CONSTRAINT Component_ak_ComponentName UNIQUE (ComponentName),
    CONSTRAINT Component_pk PRIMARY KEY  (ComponentID)
);
```

3.3. ComponentCategory (Kategorie Komponentów) - słownik kategorii służący do grupowania surowców

- CategoryID: Unikalny identyfikator kategorii.
- CategoryName: Nazwa kategorii

```
SQL
CREATE TABLE ComponentCategory (
    CategoryID int  NOT NULL,
    CategoryName varchar(100)  NOT NULL,
    CONSTRAINT ComponentCategory_ak_CatName UNIQUE (CategoryName),
    CONSTRAINT ComponentCategory_pk PRIMARY KEY  (CategoryID)
);
```

3.4. ComponentSupplyDetails - tabela łącząca informująca, który dostawca oferuje dany komponent i na jakich warunkach

- ComponentID: Identyfikator komponentu.
- SupplierID: Identyfikator dostawcy.
- ShipTime: Średni czas dostawy w dniach (kluczowe do planowania produkcji).
- ComponentPrice: Cena jednostkowa zakupu komponentu u tego dostawcy.
- ShipmentPrice: Koszt przesyłki od tego dostawcy.

```
SQL
CREATE TABLE ComponentSupplyDetails (
    ComponentID int  NOT NULL,
    SupplierID int  NOT NULL,
    ShipTime int  NOT NULL,
    ComponentPrice money  NOT NULL,
    ShipmentPrice money  NOT NULL,
    CONSTRAINT ComponentSupplyDetails_pk PRIMARY KEY  (ComponentID,SupplierID)
);
```

3.5. Country słownik krajów, wykorzystywany w adresach miast.

- CountryID: Unikalny identyfikator kraju.
- CountryName: Pełna nazwa kraju.

```
SQL
CREATE TABLE Country (
    CountryID int  NOT NULL,
    CountryName varchar(100)  NOT NULL,
    CONSTRAINT Country_ak_CountryName UNIQUE (CountryName),
    CONSTRAINT Country_pk PRIMARY KEY  (CountryID)
);
```

3.6. Customer (Klienci - Dane Podstawowe)- tabela bazowa przechowująca dane kontaktowe każdego klienta (zarówno firm, jak i osób prywatnych).

- CustomerID: Unikalny identyfikator klienta.
- Email: Adres e-mail (unikalny, z walidacją formatu).
- Phone: Numer telefonu (unikalny, z walidacją cyfr).
- Address: Ulica i numer domu/lokalu.
- PostalCode: Kod pocztowy (z walidacją formatu XX-XXX).
- CityID: Powiązanie z tabelą miast.

```
SQL
CREATE TABLE Customer (
    CustomerID int  NOT NULL,
    Email varchar(100)  NOT NULL,
    Phone varchar(15)  NOT NULL,
    Address varchar(255)  NOT NULL,
```

```
   PostalCode varchar(6)  NOT NULL,
   CityID int  NOT NULL,
   CONSTRAINT Customer_ak_Phone UNIQUE (Phone),
   CONSTRAINT Customer_ak_Email UNIQUE (Email),
   CONSTRAINT Customer_ak_Address UNIQUE (Address),
   CONSTRAINT CheckEmail CHECK (Email LIKE '%@%' AND Email NOT LIKE '@%' AND Email NOT LIKE '%@'),
   CONSTRAINT CheckPhone CHECK (Phone ~ '^[0-9]{9,}$'),
   CONSTRAINT CheckPostalCode CHECK (PostalCode LIKE '%-%' AND PostalCode NOT LIKE '-%' AND PostalCode NOT
LIKE '%-' ),
   CONSTRAINT Customer_pk PRIMARY KEY  (CustomerID)
);
```

3.7. CustomerCompany - rozszerzenie tabeli Customer dla klientów biznesowych

- CustomerID: Identyfikator klienta (taki sam jak w tabeli Customer).
- CompanyName: Nazwa firmy.
- Fax: Numer faksu.
- ContactName: Imię i nazwisko osoby kontaktowej w firmie.
- NIP: Numer Identyfikacji Podatkowej (unikalny).

```
SQL
CREATE TABLE CustomerCompany (
   CustomerID int  NOT NULL,
   CompanyName varchar(100)  NOT NULL,
   Fax varchar(10)  NOT NULL,
   ContactName varchar(100)  NOT NULL,
   NIP varchar(10)  NOT NULL,
   CONSTRAINT CustomerCompany_ak_NIP UNIQUE (NIP),
   CONSTRAINT CustomerCompany_ak_Fax UNIQUE (Fax),
   CONSTRAINT CustomerCompany_ak_CompanyName UNIQUE (CompanyName),
   CONSTRAINT CheckFax CHECK (Fax NOT LIKE '%[^0-9]%'),
   CONSTRAINT CheckNIP CHECK (NIP NOT LIKE '%[^0-9]%'),
   CONSTRAINT CustomerCompany_pk PRIMARY KEY  (CustomerID)
);
```

3.8. CustomerPerson - rozszerzenie tabeli Customer dla klientów indywidualnych.

- CustomerID: Identyfikator klienta.
- FirstName: Imię klienta.
- LastName: Nazwisko klienta.

```
SQL
CREATE TABLE CustomerPerson (
   CustomerID int  NOT NULL,
   FirstName varchar(100)  NOT NULL,
   LastName varchar(100)  NOT NULL,
   CONSTRAINT CustomerPerson_pk PRIMARY KEY  (CustomerID)
);
```

3.9. Employee - kartoteka pracowników firmy, zawierająca dane kadrowe oraz strukturę podległości.

- EmployeeID: Identyfikator pracownika.
- RoleID: Powiązanie ze stanowiskiem (EmployeeRole).
- BirthDate: Data urodzenia.
- FirstName / LastName: Imię i Nazwisko.

- HireDate: Data zatrudnienia.
- Phone: Numer telefonu służbowego/prywatnego.
- ReportsTo: Identyfikator przełożonego (relacja rekurencyjna do tej samej tabeli).
- Address / PostalCode / CityID: Dane adresowe pracownika.

```
SQL
CREATE TABLE Employee (
    EmployeeID int  NOT NULL,
    RoleID int  NOT NULL,
    BirthDate date  NOT NULL,
    FirstName varchar(100)  NOT NULL,
    LastName varchar(100)  NOT NULL,
    HireDate date  NOT NULL,
    Phone varchar(15)  NOT NULL,
    ReportsTo int  NULL,
    Address varchar(255)  NOT NULL,
    PostalCode varchar(6)  NOT NULL,
    CityID int  NOT NULL,
    CONSTRAINT Employee_ak_Phone UNIQUE (Phone),
    CONSTRAINT CheckPhone CHECK (Phone ~ '^[0-9]{9,}$'),
    CONSTRAINT CheckPostalCode CHECK (PostalCode LIKE '%-%' AND PostalCode NOT LIKE '-%' AND PostalCode NOT
LIKE '%-' ),
    CONSTRAINT CheckBirthDate CHECK (BirthDate > HireDate),
    CONSTRAINT Employee_pk PRIMARY KEY  (EmployeeID)
);
```

3.10. EmployeeRole - słownik stanowisk oraz stawek płacowych w firmie.

- RoleID: Identyfikator roli.
- RoleName: Nazwa stanowiska (np. "Monter", "Sprzedawca", "Magazynier").
- HourlyPay: Stawka godzinowa przypisana do stanowiska (ułatwia obliczanie kosztów robocizny).

```
SQL
CREATE TABLE EmployeeRole (
    RoleID int  NOT NULL,
    RoleName varchar(100)  NOT NULL,
    HourlyPay money  NOT NULL,
    CONSTRAINT EmployeeRole_ak_RoleName UNIQUE (RoleName),
    CONSTRAINT EmployeeRole_pk PRIMARY KEY  (RoleID)
);
```

3.11. OrderDetails - szczegóły zamówienia klienta – konkretne produkty (meble) i ich ilości w ramach jednego zamówienia.

- OrderID: Identyfikator nagłówka zamówienia.
- ProductID: Identyfikator zamawianego mebla.
- UnitPrice: Cena sprzedaży w momencie zakupu (gwarancja ceny dla klienta).
- Quantity: Liczba zamówionych sztuk.
- Discount: Udzielony rabat (w formacie dziesiętnym, np. 0.05).

```
SQL
CREATE TABLE OrderDetails (
    OrderID int  NOT NULL,
    ProductID int  NOT NULL,
    UnitPrice money  NOT NULL,
    Quantity int  NOT NULL,
    Discount decimal(3,2)  NOT NULL,
    CONSTRAINT OrderDetails_pk PRIMARY KEY  (OrderID,ProductID)
);
```

3.12. Orders - główna tabela transakcyjna przechowująca nagłówki zamówień sprzedażowych.

- OrderID: Numer zamówienia.
- CustomerID: Klient składający zamówienie.
- EmployeeID: Pracownik obsługujący zamówienie (sprzedawca).
- OrderDate: Data wpłynięcia zamówienia.
- PlannedShipDate: Obiecana data wysyłki.
- ShipmentDate: Faktyczna data wysyłki (wypełniana po zrealizowaniu).
- Freight: Koszt dostawy doliczany do zamówienia.
- ShipperID: Wybrana firma kurierska.

```
SQL
CREATE TABLE Orders (
    OrderID int  NOT NULL,
    CustomerID int  NOT NULL,
    EmployeeID int  NOT NULL,
    OrderDate datetime  NOT NULL,
    PlannedShipDate datetime  NOT NULL,
    ShipmentDate datetime  NULL,
    Freight money  NOT NULL,
    ShipperID int  NOT NULL,
    CONSTRAINT Orders_pk PRIMARY KEY  (OrderID)
);
```

3.13. Product - katalog oferowanych mebli (produktów gotowych).

- ProductID: Identyfikator produktu.
- ProductName: Nazwa handlowa mebla.
- ProductCategoryID: Kategoria (np. Stoły, Krzesła).
- SellingPrice: Cena katalogowa dla klienta.
- ProductionTime: Szacowany czas produkcji jednej sztuki (w minutach) - kluczowe dla algorytmów planowania.
- QuantityInStock: Ilość gotowych mebli na magazynie.
- ReservedQuantity: Ilość mebli zarezerwowana pod bieżące zamówienia.
- Discontinued: Flaga (0/1) oznaczająca wycofanie produktu ze sprzedaży.

```
SQL
```

```
CREATE TABLE Product (
    ProductID int  NOT NULL,
    ProductName varchar(100)  NOT NULL,
    ProductCategoryID int  NOT NULL,
    ProductionPrice money  NOT NULL,
    SellingPrice money  NOT NULL,
    ProductionTime int  NOT NULL,
    QuantityInStock int  NOT NULL,
    ReservedQuantity int  NOT NULL,
    Discontinued bit  NOT NULL,
    CONSTRAINT Product_ak_ProductName UNIQUE (ProductName),
    CONSTRAINT Product_pk PRIMARY KEY  (ProductID)
);
```

3.14. ProductCategory - słownik kategorii mebli ułatwiający nawigację i raportowanie sprzedaży.

- CategoryID: Identyfikator kategorii.
- CategoryName: Nazwa (np. "Sypialnia", "Biuro", "Ogród").

```
SQL
CREATE TABLE ProductCategory (
    CategoryID int  NOT NULL,
    CategoryName varchar(100)  NOT NULL,
    CONSTRAINT ProductsCategory_ak_CatName UNIQUE (CategoryName),
    CONSTRAINT ProductCategory_pk PRIMARY KEY  (CategoryID)
);
```

3.15. ProductDetails - definiuje skład produktu (Bill of Materials). Określa, jakie komponenty i w jakiej ilości są potrzebne do stworzenia jednego mebla.

- ProductID: Mebel, którego dotyczy receptura.
- ComponentID: Potrzebny komponent (np. śruba, noga stołu).
- ComponentQuantityToProduceProduct: Liczba sztuk komponentu na jeden gotowy produkt.

```
SQL
CREATE TABLE ProductDetails (
    ProductID int  NOT NULL,
    ComponentID int  NOT NULL,
    ComponentQuantityToProduceProduct int  NOT NULL,
    CONSTRAINT ProductDetails_pk PRIMARY KEY  (ProductID,ComponentID)
);
```

3.16. Production - rejestr procesu produkcji mebli. Każdy rekord to konkretne zadanie dla pracownika.

- ProductionID: Numer zlecenia produkcyjnego.
- ProductID: Co jest produkowane.
- EmployeeID: Kto produkuje (monter/stolarz).
- QuantityProduced: Wielkość partii produkcyjnej.
- ProductionPrice: Koszt produkcji

- StartDate: Data rozpoczęcia pracy.
- ScheduledDate: Planowana data zakończenia.
- EndDate: Faktyczna data zakończenia (NULL oznacza, że produkcja trwa).
- Damaged: Czy produkt uległ uszkodzeniu podczas produkcji.
- FineAssessed: Ewentualna kara finansowa dla pracownika za zniszczenie materiału.

```
SQL
CREATE TABLE Production (
    ProductionID int  NOT NULL,
    ProductID int  NOT NULL,
    EmployeeID int  NOT NULL,
    QuantityProduced int  NOT NULL,
    StartDate datetime  NOT NULL,
    ScheduledDate datetime  NOT NULL,
    EndDate datetime  NULL,
    Damaged bit  NOT NULL,
    FineAssessed money  NULL,
    CONSTRAINT Production_pk PRIMARY KEY  (ProductionID)
);
```

3.17. Shipper - baza firm kurierskich i transportowych realizujących wysyłkę mebli do klientów.

- ShipperID: Identyfikator przewoźnika.
- CompanyName: Nazwa firmy przewozowej.
- Address / Phone / CityID: Dane kontaktowe przewoźnika.

```
SQL
CREATE TABLE Shipper (
    ShipperID int  NOT NULL,
    CompanyName varchar(100)  NOT NULL,
    Address varchar(255)  NOT NULL,
    Phone varchar(15)  NOT NULL,
    Fax varchar(10)  NOT NULL,
    PostalCode varchar(6)  NOT NULL,
    CityID int  NOT NULL,
    CONSTRAINT Shipper_ak_Phone UNIQUE (Phone),
    CONSTRAINT Shipper_ak_Fax UNIQUE (Fax),
    CONSTRAINT Shipper_ak_Address UNIQUE (Address),
    CONSTRAINT Shipper_ak_CompanyName UNIQUE (CompanyName),
    CONSTRAINT CheckPhone CHECK (Phone NOT LIKE '^[0-9]{9,}$'),
    CONSTRAINT CheckFax CHECK (Fax ~ '^[0-9]{9,}$'),
    CONSTRAINT CheckPostalCode CHECK (PostalCode LIKE '%-%' AND PostalCode NOT LIKE '-%' AND PostalCode NOT
LIKE '%-' ),
    CONSTRAINT Shipper_pk PRIMARY KEY  (ShipperID)
);
```

3.18. Status - lista możliwych stanów, w jakich może znaleźć się produkcja (np. "Oczekuje", "W toku", "Zakończone", "Wstrzymane").

- StatusID: Identyfikator statusu.
- StatusName: Krótka nazwa.

- StatusDescription: Szczegółowy opis znaczenia statusu.

```
SQL
CREATE TABLE Status (
    StatusID int  NOT NULL,
    StatusName varchar(100)  NOT NULL,
    StatusDescription varchar(255)  NOT NULL,
    CONSTRAINT Status_ak_StatusName UNIQUE (StatusName),
    CONSTRAINT Status_pk PRIMARY KEY  (StatusID)
);
```

3.19. StatusDetails - historia przebiegu produkcji. Pozwala śledzić, ile czasu zlecenie spędziło w każdym etapie, co pomaga w optymalizacji procesów.

- StatusID: Jaki to był status.
- ProductionID: Którego zlecenia dotyczy.
- StartDate: Kiedy zlecenie weszło w ten status.
- EndDate: Kiedy zlecenie opuściło ten status.

```
SQL
CREATE TABLE StatusDetails (
    StatusID int  NOT NULL,
    ProductionID int  NOT NULL,
    StartDate datetime  NOT NULL,
    EndDate datetime  NULL,
    CONSTRAINT StatusDetails_pk PRIMARY KEY  (StatusID,ProductionID)
);
```

3.20. Supplier - baza firm zewnętrznych dostarczających komponenty niezbędne do produkcji.

- SupplierID: Identyfikator dostawcy.
- SupplierName: Nazwa firmy dostawczej.
- ContactName: Osoba kontaktowa u dostawcy.
- Address / Phone: Dane teleadresowe.

```
SQL
CREATE TABLE Supplier (
    SupplierID int  NOT NULL,
    SupplierName varchar(100)  NOT NULL,
    Address varchar(255)  NOT NULL,
    Phone varchar(15)  NOT NULL,
    ContactName varchar(100)  NOT NULL,
    PostalCode varchar(6)  NOT NULL,
    CityID int  NOT NULL,
    CONSTRAINT Supplier_ak_Name UNIQUE (SupplierName),
    CONSTRAINT Supplier_ak_Address UNIQUE (Address),
    CONSTRAINT Supplier_ak_Phone UNIQUE (Phone),
    CONSTRAINT CheckPhone CHECK (phone ~ '^[0-9]{9,}$'),
    CONSTRAINT CheckPostalCode CHECK (PostalCode LIKE '%-%' AND PostalCode NOT LIKE '-%' AND PostalCode NOT
LIKE '%-' ),
    CONSTRAINT Supplier_pk PRIMARY KEY  (SupplierID)
);
```

3.21. SupplyOrders - ewidencja zamówień wysyłanych od dostawców w celu uzupełnienia magazynu komponentów.

- SupplyID: Numer zamówienia zakupu.
- SupplierID: Od kogo zamawiamy.
- OrderDate / ScheduledShipDate / ShipmentDate: Daty złożenia, planowanej dostawy i faktycznej dostawy.
- Quantity: Ilość zamawianego towaru.
- ShipmentPrice: Koszt transportu dostawy.
- UnitPrice: Cena zakupu.

```
SQL
CREATE TABLE SupplyOrders (
    SupplyID int  NOT NULL,
    SupplierID int  NOT NULL,
    OrderDate datetime  NOT NULL,
    ScheduledShipDate datetime  NOT NULL,
    ShipmentDate datetime  NOT NULL,
    Quantity int  NOT NULL,
    ShipmentPrice money  NOT NULL,
    UnitPrice money  NOT NULL,
    Discount int  NOT NULL,
    CONSTRAINT SupplyOrders_pk PRIMARY KEY  (SupplyID,SupplierID)
);
```

3.22. Vat - słownik wartości podatku VAT w systemie.

- VatID: Identyfikator stawki.
- VatValue: Wartość dziesiętna podatku.

```
SQL
CREATE TABLE Vat (
    VatID int  NOT NULL,
    VatValue decimal(3,2)  NOT NULL,
    CONSTRAINT Vat_pk PRIMARY KEY  (VatID)
);
```

3.23. VatCategories - tabela wiążąca stawki VAT z kategoriami produktów w określonych przedziałach czasowych.

- VatID: Obowiązująca stawka VAT.
- CategoryID: Kategoria produktu, której dotyczy stawka.
- StartDate / EndDate: Okres obowiązywania danej stawki dla tej kategorii.

```
SQL
CREATE TABLE VatCategories (
    VatID int  NOT NULL,
    CategoryID int  NOT NULL,
    StartDate datetime  NOT NULL,
    EndDate datetime  NULL,
    CONSTRAINT VatCategories_pk PRIMARY KEY  (VatID,CategoryID)
);
```

4. Generowanie danych:

4.1 Opis danych

4.1.1 Dane Geograficzne i Słownikowe

- Country / City: Baza zawiera 3 kraje (Polska, Niemcy, Francja) oraz 30 miast (po 10 dla każdego kraju, m.in. Warszawa, Berlin, Paryż), co pozwala na symulację operacji międzynarodowych.
- Categories (ComponentCategory / ProductCategory): Słowniki kategoryzujące surowce (np. Drewno, Metal, Elektronika) oraz wyroby gotowe (Biurka, Krzesła, Akcesoria).
- Status / EmployeeRole: Definicje statusów procesów (np. "Scheduled", "Completed") oraz ról pracowników wraz ze stawkami godzinowymi (od Magazyniera po Dyrektora).
- Vat / VatCategories: Tabela stawek podatkowych (23%, 8%, 5%, 0%) oraz historia przypisania stawek VAT do kategorii produktów.

4.1.2 Podmioty Zewnętrzne i Wewnętrzne

- Employee: Rejestr 13 pracowników zawierający dane osobowe, daty zatrudnienia, strukturę podległości (kto jest czyim przełożonym) oraz przypisanie do konkretnych oddziałów (miast).
- Supplier: Lista 7 dostawców komponentów (np. "Woodex Ltd.", "IntelParts"), z pełnymi danymi teleadresowymi.
- Shipper: Lista 5 firm kurierskich obsługujących wysyłki (m.in. DHL, InPost, UPS).
- Customer / CustomerPerson / CustomerCompany: Baza 40 klientów, podzielona na klientów indywidualnych (20 rekordów) oraz firmowych (20 rekordów). Każdy klient posiada wygenerowane dane kontaktowe i adresowe.

4.1.3 Magazyn i Zaopatrzenie (Supply Chain)

- Component: Katalog 35 surowców i półproduktów (np. blaty dębowe, silniki elektryczne, kółka, ekrany dotykowe) wraz z informacją o stanie magazynowym i rezerwacjach.
- ComponentSupplyDetails: Tabela łącząca komponenty z dostawcami, definiująca ceny zakupu, koszty transportu i czas dostawy dla każdego elementu.
- SupplyOrders: Historia 50 zamówień do dostawców, wygenerowana algorytmicznie, zawierająca daty zamówienia, planowane i rzeczywiste daty dostaw oraz koszty.

4.1.4 Katalog Produktów (Manufacturing Inventory)

- Product: Lista 20 produktów końcowych (np. Biurko Gamingowe, Fotel Ergonomiczny, Tablica Interaktywna). Rekordy zawierają cenę produkcji, cenę sprzedaży, czas wytworzenia oraz stany magazynowe.
- ProductDetails (BOM): Tabela typu "Bill of Materials", definiująca receptury produktów – czyli jakie komponenty i w jakiej ilości są potrzebne do wytworzenia jednej sztuki danego produktu.

4.1.5 Sprzedaż i Logistyka (Sales & Logistics)

- Rejestr transakcji handlowych z klientami.
- Orders: Zestawienie 100 zamówień sprzedaży rozłożonych w czasie. Każde zamówienie przypisane jest do klienta, pracownika obsługującego (sprzedawcy) oraz firmy kurierskiej. Zawiera daty złożenia, planowanej wysyłki i rzeczywistej wysyłki.
- OrderDetails: Szczegóły zamówień (pozycje na fakturze), określające który produkt, w jakiej ilości i z jakim rabatem został zakupiony w ramach danego zamówienia.

4.1.6 Produkcja (Manufacturing Process)

- Production: Rejestr 50 zleceń produkcyjnych. Każdy rekord wskazuje, jaki pracownik (np. monter) wytworzył dany produkt, ile czasu to zajęło, czy produkt został uszkodzony podczas produkcji oraz ewentualne kary finansowe.
- StatusDetails: Historia zmian statusów dla zleceń produkcyjnych (umożliwia śledzenie, kiedy produkcja się rozpoczęła, a kiedy zakończyła).

4.2 Prompt

Na podstawie tego schematu [został wysłany schemat bazy danych] wygeneruj przykładowe dane do bazy danych, dane powinny być po angielsku. Stwórz historię 100 zamówień oraz 40 klientów. Liczbę produktów i komponentów pozostawiam tobie.

4.3 Skrypt do generowania

```
-- 1. COUNTRIES (3 records)
INSERT INTO Country (CountryID, CountryName) VALUES
(1, 'Poland'), (2, 'Germany'), (3, 'France');

-- 2. CITIES (30 records)
INSERT INTO City (CityID, CountryID, CityName) VALUES
(1, 1, 'Warsaw'), (2, 1, 'Krakow'), (3, 1, 'Poznan'), (4, 1, 'Wroclaw'), (5, 1, 'Gdansk'),
(6, 1, 'Lodz'), (7, 1, 'Szczecin'), (8, 1, 'Bydgoszcz'), (9, 1, 'Lublin'), (10, 1, 'Katowice'),
(11, 2, 'Berlin'), (12, 2, 'Munich'), (13, 2, 'Hamburg'), (14, 2, 'Cologne'), (15, 2, 'Frankfurt'),
(16, 2, 'Stuttgart'), (17, 2, 'Dusseldorf'), (18, 2, 'Dortmund'), (19, 2, 'Essen'), (20, 2, 'Leipzig'),
(21, 3, 'Paris'), (22, 3, 'Marseille'), (23, 3, 'Lyon'), (24, 3, 'Toulouse'), (25, 3, 'Nice'),
(26, 3, 'Nantes'), (27, 3, 'Strasbourg'), (28, 3, 'Montpellier'), (29, 3, 'Bordeaux'), (30, 3, 'Lille');
```

```
-- 3. CATEGORIES, ROLES AND STATUSES
INSERT INTO ComponentCategory (CategoryID, CategoryName) VALUES
(1, 'Wood'), (2, 'Metal'), (3, 'Plastic'), (4, 'Electronics'), (5, 'Upholstery');

INSERT INTO ProductCategory (CategoryID, CategoryName) VALUES
(1, 'Desks'), (2, 'Chairs and Armchairs'), (3, 'Presentation Accessories');

INSERT INTO Status (StatusID, StatusName, StatusDescription) VALUES
(1, 'Scheduled', 'Waiting to start'), (2, 'In Progress', 'Production is ongoing'), (3, 'Completed', 'Ready for warehousing'), (4, 'On Hold', 'Material shortage');

INSERT INTO EmployeeRole (RoleID, RoleName, HourlyPay) VALUES
(1, 'Director', 150.00), (2, 'Production Manager', 80.00), (3, 'Salesperson', 45.00), (4, 'Machine Operator', 40.00), (5, 'Warehouse Worker', 35.00);

-- 4. VAT and VAT CATEGORIES (ADDED - BRAKOWAŁO TEGO)
INSERT INTO Vat (VatID, VatValue) VALUES
(1, 0.23), (2, 0.08), (3, 0.05), (4, 0.00);

-- Assigning Standard VAT (23%) to existing product categories
INSERT INTO VatCategories (VatID, CategoryID, StartDate, EndDate) VALUES
(1, 1, '2020-01-01', NULL), -- Desks
(1, 2, '2020-01-01', NULL), -- Chairs
(1, 3, '2020-01-01', NULL); -- Accessories

-- 5. SUPPLIERS
INSERT INTO Supplier (SupplierID, SupplierName, Address, Phone, ContactName, PostalCode, CityID) VALUES
(1, 'Woodex Ltd.', '5 Forest St', '111-222-333', 'Adam Woody', '01-001', 3),
(2, 'Steel-Met', '10 Foundry St', '222-333-444', 'Thomas Steel', '02-002', 10),
(3, 'Plastic Fantastics', '3 Polymer St', '333-444-555', 'Eve Resin', '10-101', 11),
(4, 'ElectroParts GmbH', '1 Volt Strasse', '444-555-666', 'Hans Volt', '20-202', 12),
(5, 'Seats Pro', '7 Comfort St', '555-666-777', 'Mary Soft', '30-303', 4),
(6, 'Bolts & Screws', '99 Assembly St', '666-777-888', 'John Thread', '40-404', 1),
(7, 'TechDisplay Solutions', '12 Screen St', '777-888-999', 'Peter Pixel', '50-505', 21);

-- 6. SHIPPERS
INSERT INTO Shipper (ShipperID, CompanyName, Address, Phone, Fax, PostalCode, CityID) VALUES
(1, 'FastParcel', '1 Courier St', '999-111-111', '999-111', '00-001', 1),
(2, 'DHL Express', '5 Aviation St', '999-222-222', '999-222', '00-002', 11),
(3, 'DPD', '3 Transport St', '999-333-333', '999-333', '00-003', 1),
(4, 'InPost', '7 Locker St', '999-444-444', '999-444', '00-004', 2),
(5, 'UPS', '10 Global St', '999-555-555', '999-555', '00-005', 21);

-- 7. EMPLOYEES
INSERT INTO Employee (EmployeeID, RoleID, BirthDate, FirstName, LastName, HireDate, Phone, ReportsTo, Address, PostalCode, CityID) VALUES
(1, 1, '1975-05-12', 'John', 'Bossman', '2010-01-01', '500-000-001', NULL, '1 Villa St', '00-001', 1),
(2, 2, '1980-03-15', 'Anne', 'Managerial', '2012-05-01', '500-000-002', 1, '2 Factory St', '00-002', 1),
(3, 3, '1985-07-20', 'Peter', 'Salesman', '2015-03-10', '500-000-003', 1, '3 Commerce St', '00-003', 2),
(4, 3, '1990-09-12', 'Kate', 'Trader', '2018-09-01', '500-000-004', 1, '4 Market St', '00-004', 3),
(5, 3, '1988-11-05', 'Mark', 'Vendor', '2019-02-15', '500-000-005', 1, '5 Square St', '00-005', 4),
(6, 4, '1992-01-25', 'Thomas', 'Fitter', '2020-06-01', '500-000-006', 2, '6 Work St', '00-006', 5),
(7, 4, '1993-04-14', 'George', 'Assembler', '2020-07-01', '500-000-007', 2, '7 Technical St', '00-007', 6),
(8, 4, '1995-08-30', 'Ralph', 'Driller', '2021-01-10', '500-000-008', 2, '8 Mechanical St', '00-008', 7),
(9, 4, '1982-12-12', 'William', 'Grinder', '2014-11-20', '500-000-009', 2, '9 Locksmith St', '00-009', 8),
(10, 4, '1999-02-28', 'Michael', 'Painter', '2022-03-15', '500-000-010', 2, '10 Paint St', '00-010', 9),
(11, 5, '1978-06-18', 'Christopher', 'Package', '2013-08-05', '500-000-011', 2, '11 Warehouse St', '00-011', 10),
(12, 5, '1984-10-22', 'Darius', 'Forklift', '2016-12-01', '500-000-012', 2, '12 Pallet St', '00-012', 1),
(13, 3, '1991-05-30', 'Evelyn', 'Offer', '2021-05-20', '500-000-013', 1, '13 Office St', '00-013', 2);

-- 8. CUSTOMERS
WITH Numbers AS (SELECT TOP 40 ROW_NUMBER() OVER (ORDER BY (SELECT NULL)) AS N FROM sys.objects A, sys.objects B)
INSERT INTO Customer (CustomerID, Email, Phone, Address, PostalCode, CityID)
SELECT N,
```

```
    'client' + CAST(N AS VARCHAR) + '@mail.com',
    '600-000-' + RIGHT('00' + CAST(N AS VARCHAR), 3),
    'Customer Street ' + CAST(N AS VARCHAR),
    '00-' + RIGHT('00' + CAST(N AS VARCHAR), 3),
    (N % 30) + 1
FROM Numbers;

INSERT INTO CustomerPerson (CustomerID, FirstName, LastName)
SELECT CustomerID, 'FirstName' + CAST(CustomerID AS VARCHAR), 'LastName' + CAST(CustomerID AS VARCHAR)
FROM Customer WHERE CustomerID <= 20;

INSERT INTO CustomerCompany (CustomerID, CompanyName, Fax, ContactName, NIP)
SELECT CustomerID, 'Company ' + CAST(CustomerID AS VARCHAR), 'FAX-' + CAST(CustomerID AS VARCHAR), 'Contact '
+ CAST(CustomerID AS VARCHAR), '12345678' + RIGHT('00' + CAST(CustomerID AS VARCHAR), 2)
FROM Customer WHERE CustomerID > 20;

-- 9. COMPONENTS
INSERT INTO Component (ComponentID, ComponentName, ComponentCategoryID, QuantityInStock, ReservedQuantity)
VALUES
(1, 'Oak Desk Top 140x70', 1, 100, 10),
(2, 'Gaming Desk Top Carbon', 3, 50, 5),
(3, 'Adjustable Metal Leg', 2, 200, 20),
(4, 'Rubber Casters (Set of 5)', 3, 300, 30),
(5, 'Gas Lift Class 4', 2, 150, 15),
(6, '3D Armrest Left', 3, 100, 10),
(7, '3D Armrest Right', 3, 100, 10),
(8, 'TILT Mechanism', 2, 80, 5),
(9, 'Seat Upholstery Foam', 5, 120, 12),
(10, 'Black Eco-leather (Roll)', 5, 40, 4),
(11, 'Steel Chair Frame', 2, 90, 9),
(12, 'Touch Panel 55 inch', 4, 30, 2),
(13, 'Interactive Whiteboard Frame', 2, 35, 3),
(14, 'Short-throw Projector', 4, 25, 1),
(15, 'Projector Ceiling Mount', 2, 40, 0),
(16, 'Mounting Screws M6 (Pack)', 2, 500, 50),
(17, 'RGB LED Strip 2m', 4, 200, 20),
(18, 'LED Remote Control', 4, 200, 20),
(19, 'Recessed USB 3.0 Hub', 4, 80, 5),
(20, 'Aluminum Cable Grommet', 2, 150, 10),
(21, 'Chair Headrest', 5, 70, 5),
(22, 'Lumbar Pillow', 5, 70, 5),
(23, 'Conference Table Top', 1, 20, 2),
(24, 'Conference Table Leg', 2, 80, 8),
(25, 'Top Connector', 2, 100, 10),
(26, 'ISO Chair Seat', 3, 300, 20),
(27, 'ISO Chair Backrest', 3, 300, 20),
(28, 'ISO Chair Frame', 2, 300, 20),
(29, 'Desk Height Controller', 4, 60, 2),
(30, 'Electric Leg Motor', 4, 120, 4),
(31, 'Cable Management Spine', 3, 90, 5),
(32, 'Headphone Holder', 3, 150, 10),
(33, 'Cup Holder', 3, 150, 10),
(34, 'Floor Mat', 3, 80, 0),
(35, 'Power Supply 12V', 4, 100, 10);

-- Linking components to suppliers
INSERT INTO ComponentSupplyDetails (ComponentID, SupplierID, ShipTime, ComponentPrice, ShipmentPrice)
SELECT ComponentID, (ComponentID % 7) + 1, 7, CAST(ComponentID * 10.50 AS MONEY), 50.00
FROM Component;

-- 10. SUPPLY ORDERS
-- Generating supply orders based on ComponentSupplyDetails
WITH SupplyNums AS (SELECT TOP 50 ROW_NUMBER() OVER (ORDER BY (SELECT NULL)) AS N FROM sys.objects A,
sys.objects B)
INSERT INTO SupplyOrders (SupplyID, SupplierID, ComponentID, OrderDate, ScheduledShipDate, ShipmentDate, Quantity,
ShipmentPrice, UnitPrice, Discount)
```

```
SELECT
    N,
    CSD.SupplierID,
    CSD.ComponentID,
    DATEADD(day, - (N % 100 + 10), GETDATE()), -- Order date
    DATEADD(day, - (N % 100), GETDATE()),      -- Scheduled
    DATEADD(day, - (N % 100), GETDATE()),      -- Actual Ship
    (N * 5) + 10,                              -- Quantity
    CSD.ShipmentPrice,
    CSD.ComponentPrice,
    0
FROM SupplyNums SN
JOIN ComponentSupplyDetails CSD ON CSD.ComponentID = (SN.N % 35) + 1;

-- 11. PRODUCTS
INSERT INTO Product (ProductID, ProductName, ProductCategoryID, ProductionPrice, SellingPrice, ProductionTime,
QuantityInStock, ReservedQuantity, Discontinued) VALUES
(1, 'Gaming Desk PRO LED', 1, 400.00, 899.00, 4, 50, 5, 0),
(2, 'Electric Desk StandUp', 1, 800.00, 1599.00, 5, 30, 2, 0),
(3, 'Simple Office Desk', 1, 200.00, 399.00, 2, 100, 10, 0),
(4, 'Corner Manager Desk', 1, 600.00, 1299.00, 6, 20, 1, 0),
(5, 'Gaming Chair Racer X', 2, 350.00, 799.00, 3, 40, 4, 0),
(6, 'Ergonomic Office Chair', 2, 450.00, 999.00, 3, 45, 5, 0),
(7, 'Executive Leather Chair', 2, 700.00, 1899.00, 5, 15, 0, 0),
(8, 'Conference Chair ISO', 2, 50.00, 129.00, 1, 200, 20, 0),
(9, 'SmartTouch Interactive Board', 3, 2000.00, 3500.00, 10, 10, 1, 0),
(10, 'Mobile Projector Stand', 3, 150.00, 299.00, 2, 25, 0, 0),
(11, 'Modular Conference Table', 1, 500.00, 1100.00, 4, 10, 0, 0),
(12, 'Metal Under-desk Cabinet', 3, 150.00, 350.00, 2, 60, 5, 0),
(13, 'DualArm Monitor Mount', 3, 80.00, 199.00, 1, 80, 0, 0),
(14, 'AirFlow Mesh Chair', 2, 300.00, 650.00, 3, 50, 2, 0),
(15, 'Basic Computer Desk', 1, 150.00, 299.00, 2, 120, 15, 0),
(16, 'Office Footrest', 3, 40.00, 99.00, 1, 50, 0, 0),
(17, 'Acoustic Partition Wall', 3, 200.00, 499.00, 3, 30, 0, 0),
(18, 'Conference Set (Table+4Chairs)', 1, 900.00, 1999.00, 5, 5, 0, 0),
(19, 'Rotating Whiteboard', 3, 250.00, 550.00, 2, 20, 0, 0),
(20, 'Gaming Chair RGB', 2, 500.00, 1099.00, 4, 25, 2, 0);

INSERT INTO ProductDetails (ProductID, ComponentID, ComponentQuantityToProduceProduct)
SELECT ProductID, (ProductID % 35) + 1, 2 FROM Product;

-- 12. ORDERS
WITH OrderNums AS (SELECT TOP 100 ROW_NUMBER() OVER (ORDER BY (SELECT NULL)) AS N FROM sys.objects A,
sys.objects B)
INSERT INTO Orders (OrderID, CustomerID, EmployeeID, OrderDate, PlannedShipDate, ShipmentDate, Freight, ShipperID)
SELECT
    N,
    (ABS(CHECKSUM(NEWID())) % 40) + 1,
    (ABS(CHECKSUM(NEWID())) % 4) + 3,
    DATEADD(day, - (N % 365), GETDATE()),
    DATEADD(day, - (N % 365) + 7, GETDATE()),
    DATEADD(day, - (N % 365) + 5, GETDATE()),
    50.00 + (N % 50),
    (N % 5) + 1
FROM OrderNums;

INSERT INTO OrderDetails (OrderID, ProductID, UnitPrice, Quantity, Discount)
SELECT
    O.OrderID,
    P.ProductID,
    P.SellingPrice,
    (O.OrderID % 5) + 1,
    0.00
FROM Orders O
JOIN Product P ON P.ProductID = (O.OrderID % 20) + 1;
```

```
INSERT INTO OrderDetails (OrderID, ProductID, UnitPrice, Quantity, Discount)
SELECT
    O.OrderID,
    P.ProductID,
    P.SellingPrice,
    1,
    0.05
FROM Orders O
JOIN Product P ON P.ProductID = ((O.OrderID + 5) % 20) + 1
WHERE O.OrderID % 2 = 0;

-- 13. PRODUCTION
WITH ProdNums AS (SELECT TOP 50 ROW_NUMBER() OVER (ORDER BY (SELECT NULL)) AS N FROM sys.objects A,
sys.objects B)
INSERT INTO Production (ProductionID, ProductID, EmployeeID, QuantityProduced, StartDate, ScheduledDate, EndDate,
Damaged, FineAssessed)
SELECT
    N,
    (N % 20) + 1,
    (N % 5) + 6,
    (N % 10) + 5,
    DATEADD(day, -N, GETDATE()),
    DATEADD(day, -N + 2, GETDATE()),
    DATEADD(day, -N + 2, GETDATE()),
    CASE WHEN N % 10 = 0 THEN 1 ELSE 0 END,
    CASE WHEN N % 10 = 0 THEN 50.00 ELSE NULL END
FROM ProdNums;

-- 14. STATUS DETAILS
-- Adding 'Completed' status (ID 3) to all generated finished productions
INSERT INTO StatusDetails (StatusID, ProductionID, StartDate, EndDate)
SELECT
    3, -- Completed
    ProductionID,
    EndDate, -- Started status when production ended
    NULL     -- Current status
FROM Production;
```

4.4 Przemyślenia

Gemini w miarę sobie poradził poza tym, że musieliśmy ręcznie zmienić niektóre dane jak np. czasy produkcji na minuty zamiast godzin czy rozbudować listę z czego składają się produkty. W skrócie oszczędziło to dużo czasu, a wygenerowane dane są poprawne. Główną wadą jest brak kreatywności modelu, dla przykładu wygenerowane maile klientów są w postaci “client<ID>@mail.com”.

5. Widoki

Kamil Gieras:

5.1 Ujednolicona lista klientów - łączy dwie oddzielne tabele (Klienci Firmowi i Indywidualni) w jedną listę.

```
SELECT
  e.EmployeeID, e.FirstName, e.LastName, er.RoleName, e.BirthDate, e.Phone, co.CountryName, c.CityName,
e.Address


From Employee as e
JOIN EmployeeRole as er
```

```
  ON e.RoleID = er.RoleID
Join City as c
  ON e.CityID = c.CityID
JOIN Country as co
  ON c.CountryID = co.CountryID
```

5.2 Podsumowanie zamówień (z klientami, z kosztami i VAT, wysyłka itd) - pobiera zamówienia, podpina nazwę klienta i wylicza kwoty finansowe.

```
SELECT o.OrderID,
      o.OrderDate,
      CASE
          WHEN cp.FirstName IS NOT NULL THEN cp.FirstName + ' ' + cp.LastName
          ELSE cc.CompanyName
          END
AS CustomerName,
      s.CompanyName
AS ShipperName,
      o.ShipmentDate,
      SUM(od.Quantity * od.UnitPrice * (1.0 - od.Discount))
AS OrderNetValue,
      SUM((od.Quantity * od.UnitPrice * (1.0 - od.Discount)) * v.VatValue)
AS OrderVatAmount,
      o.Freight,
      SUM((od.Quantity * od.UnitPrice * (1.0 - od.Discount)) * (1.0 + v.VatValue)) +
o.Freight AS OrderGrandTotal
FROM dbo.Orders o
        JOIN dbo.Customer c ON o.CustomerID = c.CustomerID
        LEFT JOIN dbo.CustomerPerson cp ON c.CustomerID = cp.CustomerID
        LEFT JOIN dbo.CustomerCompany cc ON c.CustomerID = cc.CustomerID
        JOIN dbo.Shipper s ON o.ShipperID = s.ShipperID
        JOIN dbo.OrderDetails od ON o.OrderID = od.OrderID
        JOIN dbo.Product p ON od.ProductID = p.ProductID
        JOIN dbo.ProductCategory pc ON p.ProductCategoryID = pc.CategoryID
        JOIN dbo.VatCategories vc ON pc.CategoryID = vc.CategoryID
   AND o.OrderDate >= vc.StartDate
   AND o.OrderDate <= ISNULL(vc.EndDate, '9999-12-31')
   AND o.ShipmentDate IS NOT NULL
        JOIN dbo.Vat v ON vc.VatID = v.VatID

GROUP BY o.OrderID,
        o.OrderDate,
        o.ShipmentDate,
        o.Freight,
        s.CompanyName,
        cp.FirstName, cp.LastName, cc.CompanyName
```

5.3 Brakujące komponenty - filtruje magazyn części, pokazując tylko te pozycje, gdzie stan dostępny do sprzedaży jest mniejszy niż 15.

```
select *
from Component
where QuantityInStock-ReservedQuantity < 15
```

5.4 Brakujące produkty - filtruje magazyn produktu, pokazując tylko te pozycje, gdzie stan dostępny do sprzedaży jest mniejszy niż 15.

```
select *
from Product
where QuantityInStock - ReservedQuantity < 15
```

5.5 **Wydatki miesięczne, kwartalne, roczne** - grupuje wszystkie koszty z tabeli wydatków według daty (miesięcznie,kwartalnie,rocznie)

po kategori produktu:

```
SELECT TOP 100 PERCENT YEAR(ph.StartDate)              AS [Year],
                       DATEPART(QUARTER, ph.StartDate) AS [Quarter],
                       MONTH(ph.StartDate)             AS [Month],
                       ProductCategory.CategoryName    AS [Category],
                       SUM(ph.ProductionPrice)         AS [Category Spends]
   FROM ProductionHistory ph
            INNER JOIN Product ON Product.ProductID = ph.ProductID
            INNER JOIN ProductCategory ON Product.ProductCategoryID =
ProductCategory.CategoryID
   GROUP BY YEAR(ph.StartDate),
            MONTH(ph.StartDate),
            DATEPART(QUARTER, ph.StartDate),
            ProductCategory.CategoryName
   ORDER BY [Year] DESC,
            [Month] DESC,
            [Category] ASC
```

po produktach:

```
SELECT TOP 100 PERCENT YEAR(ph.StartDate)              AS [Year],
                       DATEPART(QUARTER, ph.StartDate) AS [Quarter],
                       MONTH(ph.StartDate)             AS [Month],
                       Product.ProductName             AS [Product Name],
                       SUM(ph.ProductionPrice)         AS [Category Spends]
   FROM ProductionHistory ph
            INNER JOIN Product ON Product.ProductID = ph.ProductID
            INNER JOIN ProductCategory ON Product.ProductCategoryID =
ProductCategory.CategoryID
   GROUP BY YEAR(ph.StartDate),
            MONTH(ph.StartDate),
            DATEPART(QUARTER, ph.StartDate),
            Product.ProductName
   ORDER BY [Year] DESC,
            [Month] DESC,
            [Product Name] ASC
```

Łukasz Lewczak

5.6 **BOM** - lista produktów i z czego każdy produkt powstaje

```
create view [dbo].[ProductBillOfMaterials]
        as
        select pd.ProductID, p.ProductName, pd.ComponentID, c.ComponentName,
        pd.ComponentQuantityToProduceProduct
        from ProductDetails pd
        inner join Product p
        on pd.ProductID = p.ProductID
        inner join Component c
        on c.ComponentID = pd.ComponentID
        GO
```

5.7 **Produkty** z ich cenami (z uwzględnionym aktualnym VAT)

```
create view [dbo].[ProductPriceAfterVat]
```

```
as
	select p.ProductID, p.ProductName, cast(p.SellingPrice*(1+isnull(v.VatValue, 0)) as decimal(10,2)) as Price
	from Product p
	left join VatCategories vc
	on p.ProductCategoryID = vc.CategoryID and vc.EndDate is null
	left join Vat v
	on v.VatID = vc.VatID
GO
```

5.8 Produkcja - historia z ich statusami

```
create view [dbo].[ProductionHistory]
as
	select sd.ProductionID, prdc.ProductID, prdc.ProductName, e.EmployeeID, concat(e.FirstName ,' ', e.LastName) as
	EmployeeName, s.StatusName, sd.StartDate, sd.EndDate, DATEDIFF(MINUTE, sd.StartDate, sd.EndDate) as
	DurationInMinutes
	from StatusDetails sd
	inner join Production p
	on sd.ProductionID = p.ProductionID
	inner join Product prdc
	on prdc.ProductID = p.ProductID
	inner join Status s
	on s.StatusID = sd.StatusID
	left join Employee e
	on e.EmployeeID = p.EmployeeID
GO
```

5.9 Widok generujący grafik

```
CREATE VIEW [dbo].[EmployeeScheduleTemplate]
AS
WITH DateGenerator AS (
   SELECT CAST(GETDATE() + 1 AS DATE) AS FutureDate
   UNION ALL
   SELECT DATEADD(DAY, 1, FutureDate)
   FROM DateGenerator
   WHERE DATEADD(DAY, 1, FutureDate) <= DATEADD(DAY, 30, GETDATE())
),
AllAssignments AS (
   SELECT
      dg.FutureDate,
      e.EmployeeID,
      CONCAT(e.FirstName, ' ', e.LastName) AS EmployeeName,
      ROW_NUMBER() OVER(
         PARTITION BY dg.FutureDate
         ORDER BY NEWID()
      ) as RandomNumber
   FROM DateGenerator dg
   CROSS JOIN dbo.Employee e
)
SELECT
   FutureDate,
   EmployeeID,
   EmployeeName
FROM AllAssignments
WHERE RandomNumber <= 10;
GO
```

5.10 Raport dotyczący planu wytworzenia poszczególnych produktów w różnych przedziałach czasu.

```
create VIEW [dbo].[ProductionPlanRaport] AS
```

```
SELECT
    pc.CategoryName AS Category,
    p.ProductName AS ProductName,
    year(prod.ScheduledDate) AS PlannedYear,
    month(prod.ScheduledDate) AS PlannedMonth,
    COUNT(prod.ProductionID) AS NumberOfProductionOrders,
    SUM(prod.QuantityProduced) AS PlannedQuantity,
    SUM(prod.QuantityProduced * p.ProductionTime) AS EstimatedTimeLoad
FROM
    Production prod
JOIN
    Product p ON prod.ProductID = p.ProductID
JOIN
    ProductCategory pc ON p.ProductCategoryID = pc.CategoryID
GROUP BY
    pc.CategoryName,
    p.ProductName,
    year(prod.ScheduledDate),
    month(prod.ScheduledDate)
GO
```

5.11 Raporty dla kadry zarządczej dotyczące sprzedaży grup produktów w poszczególnych miesiącach oraz kosztów produkcji produktów

```
create VIEW [dbo].[ProductCategorySale] AS
WITH
ComponentAvgPrice AS (
    SELECT
        ComponentID,
        AVG(ComponentPrice) AS Price
    FROM ComponentSupplyDetails
    GROUP BY ComponentID
),
ProductUnitCost AS (
    SELECT
        pd.ProductID,
        SUM(pd.ComponentQuantityToProduceProduct * cap.Price) AS UnitMaterialCost
    FROM ProductDetails pd
    JOIN ComponentAvgPrice cap ON pd.ComponentID = cap.ComponentID
    GROUP BY pd.ProductID
),
UnifiedData AS (
    --Sprzedaż
    SELECT
        YEAR(o.OrderDate) AS ReportYear,
        MONTH(o.OrderDate) AS ReportMonth,
        DATEPART(week, o.OrderDate) AS ReportWeek,
        pc.CategoryName,
        (od.Quantity * od.UnitPrice) AS SalesVal,
        0.00 AS ProductionVal
    FROM Orders o
    JOIN OrderDetails od ON o.OrderID = od.OrderID
    JOIN Product p ON od.ProductID = p.ProductID
    JOIN ProductCategory pc ON p.ProductCategoryID = pc.CategoryID

    UNION ALL

    --Produkcja
    SELECT
        YEAR(pr.StartDate) AS ReportYear,
        MONTH(pr.StartDate) AS ReportMonth,
        DATEPART(week, pr.StartDate) AS ReportWeek,
        pc.CategoryName,
        0.00 AS SalesVal,
        (pr.QuantityProduced * ISNULL(puc.UnitMaterialCost, 0)) AS ProductionVal
```

```
    FROM Production pr
    JOIN Product p ON pr.ProductID = p.ProductID
    JOIN ProductCategory pc ON p.ProductCategoryID = pc.CategoryID
    LEFT JOIN ProductUnitCost puc ON p.ProductID = puc.ProductID
)

SELECT
    ReportYear AS [Year],
    ReportMonth AS [Month],
    ReportWeek AS [Week],
    CategoryName AS [Product Category],
    SUM(SalesVal) AS [Total Sales],
    SUM(ProductionVal) AS [Production Cost]
FROM UnifiedData
GROUP BY
    ReportYear,
    ReportMonth,
    ReportWeek,
    CategoryName;
GO
```

Stanisław Wawrylak

5.12 Historia zamowien polproduktow z pelnymi nazwami komponentow i dostawcow

```
CREATE VIEW [dbo].[HalfProductsHistory]
AS
SELECT
        so.SupplyID, so.OrderDate, so.ScheduledShipDate, so.ShipmentDate, ((so.Quantity*so.UnitPrice)*(1 - so.Discount) +
so.ShipmentPrice) as Price,
        c.ComponentName, cc.CategoryName,
        s.SupplierName, s.ContactName, s.Phone

FROM SupplyOrders as so
LEFT JOIN Supplier as s
        ON so.SupplierID = s.SupplierID
LEFT JOIN Component as c
        On so.ComponentID = c.ComponentID
Join ComponentCategory as cc
        ON c.ComponentCategoryID = cc.CategoryID
GO
```

5.13 ShippersInfo - informacje o kurierach (z krajami)

```
CREATE VIEW [dbo].[ShippersInfo]
AS
SELECT
        s.CompanyName, s.Address, s.Phone, co.CountryName, c.CityName


FROM Shipper as s
JOIN City as c
        ON s.CityID = c.CityID
Join Country as co
        On co.CountryID = c.CountryID
GO
```

5.14 SuppliersInfo - informacje o dostawcach z krajami

```
CREATE VIEW [dbo].[SuppliersInfo]
AS
SELECT
	s.SupplierName, s.ContactName, s.Address, s.Phone, co.CountryName, c.CityName


FROM Supplier as s
JOIN City as c
	ON s.CityID = c.CityID
Join Country as co
	On co.CountryID = c.CountryID
GO
```

5.15 Pracownicy pełne dane (wypłata, kraj itp)

```
CREATE VIEW [dbo].[EmployeesInfo]
AS
SELECT
	e.EmployeeID, e.FirstName, e.LastName, er.RoleName, e.BirthDate, e.Phone, co.CountryName, c.CityName,
e.Address


From Employee as e
JOIN EmployeeRole as er
	ON e.RoleID = er.RoleID
Join City as c
	ON e.CityID = c.CityID
JOIN Country as co
	ON c.CountryID = co.CountryID
GO
```

5.16 CurrentProductAvailableInStock - Raport o aktualnym stanie magazynu wraz z rezerwacją

```
ALTER   VIEW [dbo].[CurrentProductAvailableInStock]
AS
SELECT
   ProductID,
   ProductName,
   QuantityInStock,
   ReservedQuantity,
   CASE
      WHEN (QuantityInStock - ReservedQuantity) < 0
      THEN 0
      ELSE (QuantityInStock - ReservedQuantity)
   END AS AvailableForSale
FROM Product;
GO
```

5.17 CurrentProduction - Raport o aktualnym stanie produkcji

```
ALTER VIEW [dbo].[CurrentProduction]
AS
SELECT sd.ProductionID, pr.ProductID, pr.ProductName, e.EmployeeID, concat(e.FirstName ,' ', e.LastName) as
EmployeeName, s.StatusName, p.StartDate
FROM StatusDetails sd
INNER JOIN Production p
```

```
        ON sd.ProductionID = p.ProductionID
INNER JOIN Product pr
        ON pr.ProductID = p.ProductID
INNER JOIN Status s
        ON s.StatusID = sd.StatusID
LEFT JOIN Employee e
        ON e.EmployeeID = p.EmployeeID
WHERE sd.EndDate is null and s.StatusName != 'Completed'
GO
```

5.18 CurrentProductAndProductionStateRaport - Raport o aktualnym stanie magazynowym wraz z uwzglednieniem produkcji

```
ALTER VIEW [dbo].[CurrentProductAndProductionStateRaport] AS
SELECT
        p.ProductID, p.ProductName, p.QuantityInStock, p.ReservedQuantity,
   ISNULL(SUM(pr.QuantityProduced), 0) AS PlannedToProduce,
   CASE
      WHEN (QuantityInStock - ReservedQuantity) < 0
      THEN 0
      ELSE (QuantityInStock - ReservedQuantity)
   END AS AvailableNow,
   CASE
      WHEN (p.QuantityInStock - p.ReservedQuantity + ISNULL(SUM(pr.QuantityProduced), 0)) < 0
      THEN 0
      ELSE (p.QuantityInStock - p.ReservedQuantity + ISNULL(SUM(pr.QuantityProduced), 0))
   END AS AvailableAfterProduction


FROM Product as p

LEFT JOIN Production as pr
        ON pr.ProductID = p.ProductID AND pr.EndDate is null

GROUP BY
   p.ProductID,
   p.ProductName,
   p.QuantityInStock,
   p.ReservedQuantity;
GO
```

6. PROCEDURY

Łukasz Lewczak

6.1 Składanie zamówienia

```
create procedure [dbo].[usp_CreateOrder]
        @p_CustomerID int,
        @p_EmployeeID int,
        @p_OrderDate datetime = null,
        @p_Freight money,
        @p_ShipperID int,
        @p_ProductsList dbo.ListProductQuantity readonly
as
begin
```

```
	set nocount on;
	set xact_abort on;
	begin transaction;

	declare @v_ProductID int
	declare @v_MissingQuantity int
	declare @table_PlannedDate table (PlannedDate datetime);
	declare @v_PlannedDate datetime

	if @p_OrderDate is null set @p_OrderDate = getdate();
	set @v_PlannedDate = @p_OrderDate;
	declare @v_ResultPlannedDate datetime = @p_OrderDate;

	declare @v_RandomEmployeeID int;

	declare cursor_missing_products cursor local fast_forward for
	select p.ProductID, abs(p.QuantityInStock - pl.Quantity) as MissingQuantity
	from Product p
	inner join @p_ProductsList pl
	on p.ProductID = pl.ProductID
	where p.QuantityInStock - pl.Quantity < 0

	open cursor_missing_products

	fetch next from cursor_missing_products
	into @v_ProductID, @v_MissingQuantity

	while @@FETCH_STATUS = 0
	begin
		delete from @table_PlannedDate;
		set @v_RandomEmployeeID = (ABS(CHECKSUM(NEWID())) % 12) + 2;

		insert into @table_PlannedDate
		exec dbo.usp_StartProduction
			@p_ProductID = @v_ProductID,
			@p_QuantityToProduce = @v_MissingQuantity,
			@p_EmployeeID = @v_RandomEmployeeID;

		select top 1 @v_ResultPlannedDate = PlannedDate from @table_PlannedDate;

		if @v_ResultPlannedDate > @v_PlannedDate
    begin
      set @v_PlannedDate = @v_ResultPlannedDate;
    end

		fetch next from cursor_missing_products into @v_ProductID, @v_MissingQuantity;
	end
	close cursor_missing_products;
	deallocate cursor_missing_products;

	update p
	set p.QuantityInStock = CASE
                                WHEN p.QuantityInStock < pl.Quantity THEN
0
                                ELSE p.QuantityInStock - pl.Quantity
                            END, p.ReservedQuantity = p.ReservedQuantity +
pl.Quantity
	from Product p
	inner join @p_ProductsList pl
	on p.ProductID = pl.ProductID

	SET @v_PlannedDate = DATEADD(day, 3, @v_PlannedDate);

	insert into dbo.Orders (CustomerID, EmployeeID, OrderDate,PlannedShipDate, Freight, ShipperID)
	values (@p_CustomerID, @p_EmployeeID, @p_OrderDate, @v_PlannedDate, @p_Freight, @p_ShipperID)
```

```
        declare @NewOrderID int = SCOPE_IDENTITY();

        insert into OrderDetails (OrderID, ProductID, UnitPrice, Quantity, Discount)
    select @NewOrderID, pl.ProductID, p.SellingPrice, pl.Quantity, (ABS(CHECKSUM(NEWID())) % 10) / 10.0
    from @p_ProductsList pl
        inner join Product p
        on p.ProductID = pl.ProductID

        commit transaction;
end
```

6.2 Dodanie nowego klienta indywidualnego

```
create procedure [dbo].[usp_AddPersonCustomer]
        @p_Email varchar(100),
        @p_Phone varchar(15),
        @p_Addres varchar(255),
        @p_PostalCode varchar(6),
        @p_CityID int,
        @p_FirstName varchar(100),
        @p_LastName varchar(100)
as
begin
        set nocount on;
        set xact_abort on;
        begin transaction;

                IF NOT EXISTS (SELECT 1 FROM City WHERE CityID = @p_CityID)
                BEGIN
                        RAISERROR('Błąd: Podane miasto (ID: %d) nie istnieje.', 16, 1, @p_CityID);
                        RETURN;
                END

                IF @p_Email NOT LIKE '%@%.%'
                BEGIN
                        RAISERROR('Błąd: Niepoprawny format adresu email.', 16, 1);
                        RETURN;
                END

                insert into Customer (Email, Phone, Address, PostalCode, CityID)
                values (@p_Email, @p_Phone, @p_Addres, @p_PostalCode, @p_CityID)

                declare @NewCustomerID int = scope_identity();

                insert into CustomerPerson(CustomerID, FirstName, LastName)
                values (@NewCustomerID, @p_FirstName, @p_LastName)

        commit transaction;
end
```

6.3 Dodanie nowej firmy

```
create procedure [dbo].[usp_AddCompanyCustomer]
        @p_Email varchar(100),
        @p_Phone varchar(15),
        @p_Addres varchar(255),
        @p_PostalCode varchar(6),
        @p_CityID int,
        @p_CompanyName varchar(100),
        @p_Fax varchar(10),
        @p_ContactName varchar(100),
        @p_NIP varchar(10)
as
begin
```

```
	set nocount on;
	set xact_abort on;
	begin transaction;

		IF NOT EXISTS (SELECT 1 FROM City WHERE CityID = @p_CityID)
		BEGIN
			RAISERROR('Błąd: Podane miasto (ID: %d) nie istnieje.', 16, 1, @p_CityID);
			RETURN;
		END

		IF EXISTS (SELECT 1 FROM CustomerCompany WHERE NIP = @p_NIP)
		BEGIN
			RAISERROR('Błąd: Firma o podanym numerze NIP (%s) już istnieje w bazie.', 16, 1, @p_NIP);
			RETURN;
		END

		IF LEN(@p_NIP) <> 10 OR @p_NIP LIKE '%[^0-9]%'
		BEGIN
			RAISERROR('Błąd: NIP musi składać się dokładnie z 10 cyfr.', 16, 1);
			RETURN;
		END

		IF LEN(@p_FAX) <> 10 OR @p_FAX LIKE '%[^0-9]%'
		BEGIN
			RAISERROR('Błąd: FAX musi składać się dokładnie z 10 cyfr.', 16, 1);
			RETURN;
		END

		IF @p_Email NOT LIKE '%@%.%'
		BEGIN
			RAISERROR('Błąd: Niepoprawny format adresu email.', 16, 1);
			RETURN;
		END

		insert into Customer (Email, Phone, Address, PostalCode, CityID)
		values (@p_Email, @p_Phone, @p_Addres, @p_PostalCode, @p_CityID)

		declare @NewCustomerID int = scope_identity();

		insert into CustomerCompany (CustomerID, CompanyName, Fax, ContactName, NIP)
		values (@NewCustomerID, @p_CompanyName, @p_Fax, @p_ContactName, @p_NIP)

	commit transaction;
end
```

6.4 Aktualizacja vatu

```
create procedure [dbo].[usp_CreateVat]
	@p_VatValue decimal(3,2),
	@p_CategoryID int
as
begin
	set nocount on;
	set xact_abort on;

	declare @v_VatID int;
	declare @CurrentVatID int;

	begin transaction;

		select @v_VatID = VatID
		from Vat
		where VatValue = @p_VatValue
```

```
                    if @v_VatID is null
                    begin
                            insert into Vat (VatValue)
                            values (@p_VatValue)
                            set @v_VatID = SCOPE_IDENTITY();
                    end

                    select top 1 @CurrentVatID = VatID
                    from VatCategories
                    where CategoryID = @p_CategoryID
                    order by StartDate desc;

                    if @CurrentVatID is null or @CurrentVatID <> @v_VatID
                    begin
                            update VatCategories
                            set EndDate = GETDATE()
                            where CategoryID = @p_CategoryID and EndDate is null;

                            insert into VatCategories (VatID, CategoryID, StartDate)
                            values (@v_VatID, @p_CategoryID, GETDATE());
                    end

            commit transaction;
end
```

Stanisław Wawrylak:

6.5 CreateSupplyOrder - dodanie zamowienia na czesci

```
CREATE OR ALTER   PROCEDURE [dbo].[usp_CreateSupplyOrder]
   @p_ComponentID INT,
   @p_Quantity INT,
   @p_Discount INT = 0
AS
BEGIN
   SET NOCOUNT ON;

   DECLARE
      @v_SupplierID INT,
      @v_ComponentPrice MONEY,
      @v_ShipmentPrice MONEY,
      @v_ShipTime INT,
      @v_ScheduledShipDate DATETIME,
      @v_SupplyId INT;

   SELECT
      @v_SupplierID = SupplierID,
      @v_ComponentPrice = ComponentPrice,
      @v_ShipmentPrice = ShipmentPrice,
      @v_ShipTime = ShipTime
   FROM ComponentSupplyDetails
   WHERE ComponentID = @p_ComponentID;



   IF NOT EXISTS (SELECT 1 FROM Component WHERE ComponentID = @p_ComponentID)
      RAISERROR ('Błąd: Komponent nie istnieje.', 16, 1);

   IF @v_SupplierID IS NULL
      RAISERROR ('Brak danych dostawczych dla komponentu.', 16, 1);

   SET @v_ScheduledShipDate = DATEADD(day, @v_ShipTime, GETDATE());

   SELECT @v_SupplyID = ISNULL(MAX(SupplyID), 0) + 1 FROM SupplyOrders;
```

```
    INSERT INTO SupplyOrders (
        SupplyID,
        ComponentID,
        SupplierID,
        OrderDate,
        ScheduledShipDate,
        Quantity,
        UnitPrice,
        ShipmentPrice,
        ShipmentDate,
        Discount
    )
    VALUES (
        @v_SupplyID,
        @p_ComponentID,
        @v_SupplierID,
        GETDATE(),
        @v_ScheduledShipDate,
        @p_Quantity,
        @v_ComponentPrice,
        @v_ShipmentPrice,
        NULL,
        @p_Discount
    );

END;
```

6.6 CompleteSupplyOrder - zakończenie zamowienia na czesci i dodanie do magazynu

```
CREATE OR ALTER   PROCEDURE [dbo].[usp_CompleteSupplyOrder]
    @p_SupplyID INT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE
        @v_ComponentID INT,
        @v_Quantity INT;

    SELECT
        @v_ComponentID = ComponentID,
        @v_Quantity = Quantity
    FROM SupplyOrders
    WHERE SupplyID = @p_SupplyID;

    UPDATE SupplyOrders
        SET ShipmentDate = GETDATE()
    WHERE SupplyID = @p_SupplyID;

    UPDATE Component
        SET QuantityInStock = QuantityInStock + @v_Quantity
    WHERE ComponentID = @v_ComponentID;
END;
```

6.7 UpdateProductionStatus - zmiana statusu produkcji

```
CREATE OR ALTER   PROCEDURE [dbo].[usp_UpdateProductionStatus]
    @p_ProductionID INT,
    @p_NewStatusID INT
AS
BEGIN
```

```
    SET NOCOUNT ON;

    IF NOT EXISTS (SELECT 1 FROM Production WHERE ProductionID = @p_ProductionID)
        RAISERROR('Błąd: zlecenie produkcji o ID %d nie istnieje.', 16, 1, @p_ProductionID);

    DECLARE @v_OldStatusID INT;

    SELECT TOP 1 @v_OldStatusID = StatusID
    FROM StatusDetails
    WHERE ProductionID = @p_ProductionID
    ORDER BY StartDate DESC;

    IF @v_OldStatusID IS NOT NULL
    BEGIN
        UPDATE StatusDetails
        SET EndDate = GETDATE()
        WHERE ProductionID = @p_ProductionID
          AND StatusID = @v_OldStatusID
    END

    INSERT INTO StatusDetails (ProductionID, StatusID, StartDate, EndDate)
    VALUES (@p_ProductionID, @p_NewStatusID, GETDATE(), NULL)

END;
```

Kamil Gieras

6.8 **Dodanie nowego produktu** - rejestracja nowego wyrobu gotowego w systemie sprzedaży.

```
CREATE PROCEDURE usp_AddProduct
    @p_product_name varchar(100),
    @p_category_id int,
    @p_prod_price money,
    @p_sell_price money,
    @p_prod_time int,
    @p_qty_stock int,
    @Components ComponentListType READONLY
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @v_new_prod_id int;

    IF NOT EXISTS (SELECT 1 FROM ProductCategory WHERE CategoryID = @p_category_id)
        BEGIN
            RAISERROR('Błąd: Kategoria nie istnieje!', 16, 1);
        END

    SELECT @v_new_prod_id = ISNULL(MAX(ProductID), 0) + 1 FROM Product;

    BEGIN TRANSACTION
        BEGIN TRY
            INSERT INTO Product (
                ProductID, ProductName, ProductCategoryID, ProductionPrice,
                SellingPrice, ProductionTime, QuantityInStock, ReservedQuantity,
Discontinued
            )
            VALUES (
                        @v_new_prod_id, @p_product_name, @p_category_id, @p_prod_price,
                        @p_sell_price, @p_prod_time, @p_qty_stock, 0, 0
                    );

            INSERT INTO ProductDetails (ProductID, ComponentID,
ComponentQuantityToProduceProduct)
```

```
            SELECT @v_new_prod_id, ComponentID, QuantityNeeded
            FROM @Components;

            IF @@ROWCOUNT = 0
                BEGIN
                    PRINT 'Ostrzeżenie: Dodano produkt bez składników!';
                END

            COMMIT TRANSACTION;
            PRINT 'Sukces: Produkt i jego receptura zostały zapisane.';
        END TRY
        BEGIN CATCH
            ROLLBACK TRANSACTION;
            DECLARE @Msg nvarchar(4000) = ERROR_MESSAGE();
            RAISERROR(@Msg, 16, 1);
        END CATCH
END;
go
```

6.9 Dodanie nowej części - wprowadzenie do bazy nowego surowca lub komponentu niezbędnego do produkcji.

```
CREATE PROCEDURE usp_AddComponent @p_comp_name varchar(100),
                                  @p_category_id int,
                                  @p_qty_initial int,
                                  @p_supplier_id int,
                                  @p_price money,
                                  @p_ship_time int,
                                  @p_ship_price money
AS
BEGIN
   SET NOCOUNT ON;

   DECLARE @v_new_comp_id int;

   IF NOT EXISTS (SELECT 1 FROM ComponentCategory WHERE CategoryID = @p_category_id)
       BEGIN
           RAISERROR ('Błąd: Podana kategoria (ID: %d) nie istnieje!', 16, 1,
@p_category_id);
       END

   IF NOT EXISTS (SELECT 1 FROM Supplier WHERE SupplierID = @p_supplier_id)
       BEGIN
           RAISERROR ('Błąd: Podany dostawca (ID: %d) nie istnieje!', 16, 1,
@p_supplier_id);
       END

   IF EXISTS (SELECT 1 FROM Component WHERE ComponentName = @p_comp_name)
       BEGIN
           RAISERROR ('Błąd: Komponent "%s" już istnieje w bazie!', 16, 1, @p_comp_name);
       END

   SELECT @v_new_comp_id = ISNULL(MAX(ComponentID), 0) + 1 FROM Component;

   BEGIN TRANSACTION
       BEGIN TRY
           INSERT INTO Component (ComponentID, ComponentName, ComponentCategoryID,
QuantityInStock, ReservedQuantity)
           VALUES (@v_new_comp_id, @p_comp_name, @p_category_id, @p_qty_initial, 0);
           INSERT INTO ComponentSupplyDetails (ComponentID, SupplierID, ShipTime,
ComponentPrice, ShipmentPrice)
           VALUES (@v_new_comp_id, @p_supplier_id, @p_ship_time, @p_price, @p_ship_price);
           COMMIT TRANSACTION;
           PRINT 'Sukces: Komponent został dodany poprawnie.';
       END TRY
       BEGIN CATCH
```

```
            ROLLBACK TRANSACTION;
            DECLARE @ErrMsg nvarchar(4000) = ERROR_MESSAGE();
            RAISERROR (@ErrMsg, 16, 1);
        END CATCH
END;
go
```

6.10 Rozpoczęcie produkcji - rozpoczęcie produkcji produktu i określenie jego statusu. W przypadku zbyt małej ilości części, status zostaje ustawiony na **4**, a przewidywany czas to suma czasu produkcji oraz najpóźniejszego terminu dostawy brakujących elementów. W przypadku, gdy wszystkie części są dostępne, czas oczekiwania obejmuje wyłącznie sam montaż.

```
CREATE PROCEDURE [dbo].[usp_StartProduction] @p_ProductID int,
                                @p_QuantityToProduce int,
                                @p_EmployeeID int
AS
BEGIN
   SET NOCOUNT ON;

   DECLARE @v_NewProductionID int;
   DECLARE @v_ProductionTimeDays int;
   DECLARE @v_ScheduledDate datetime;
   DECLARE @v_StatusID int;
   DECLARE @v_MissingPartsCount int;
   DECLARE @v_MaxDelayDays int = 0;
   DECLARE @v_HourlyRate decimal(18, 2); -- Stawka pracownika
   DECLARE @v_TotalLaborCost decimal(18, 2); -- Koszt robocizny całościowy
   DECLARE @v_UnitMaterialCost decimal(18, 2);-- Koszt części na 1 produkt
   DECLARE @v_TotalMaterialCost decimal(18, 2);-- Koszt części na całe zamówienie
   DECLARE @v_CalculatedPrice decimal(18, 2);
   -- Ostateczna cena produkcji

   -- 1. WALIDACJA
   IF NOT EXISTS (SELECT 1 FROM Product WHERE ProductID = @p_ProductID)
      BEGIN
         RAISERROR ('Błąd: Produkt o ID %d nie istnieje.', 16, 1, @p_ProductID);
      END

   IF NOT EXISTS (SELECT 1 FROM Employee WHERE EmployeeID = @p_EmployeeID)
      BEGIN
         RAISERROR ('Błąd: Pracownik o ID %d nie istnieje.', 16, 1, @p_EmployeeID);
      END

   SELECT @v_ProductionTimeDays = ProductionTime FROM Product WHERE ProductID = @p_ProductID;
   SELECT @v_HourlyRate = er.HourlyPay
   FROM Employee e
         JOIN EmployeeRole er ON e.RoleID = er.RoleID
   WHERE e.EmployeeID = @p_EmployeeID;

   SET @v_TotalLaborCost = (@v_ProductionTimeDays / 60) * @v_HourlyRate;

   SELECT @v_UnitMaterialCost = SUM(NajtanszeCeny.MinPrice * pd.ComponentQuantityToProduceProduct)
   FROM ProductDetails pd
         JOIN (SELECT ComponentID, MIN(ComponentPrice) AS MinPrice
               FROM ComponentSupplyDetails
               GROUP BY ComponentID) AS NajtanszeCeny ON pd.ComponentID = NajtanszeCeny.ComponentID
   WHERE pd.ProductID = @p_ProductID;

   SET @v_TotalMaterialCost = @v_UnitMaterialCost * @p_QuantityToProduce;
   SET @v_CalculatedPrice = @v_TotalLaborCost + @v_TotalMaterialCost;
```

```
DECLARE @MissingComponents TABLE
                    (
                       ShipTime int
                    );

INSERT INTO @MissingComponents (ShipTime)
SELECT (SELECT MIN(ShipTime) FROM ComponentSupplyDetails csd WHERE csd.ComponentID = pd.ComponentID)
FROM ProductDetails pd
      JOIN Component c ON pd.ComponentID = c.ComponentID
WHERE pd.ProductID = @p_ProductID
 AND c.QuantityInStock < (pd.ComponentQuantityToProduceProduct * @p_QuantityToProduce);

SELECT @v_MissingPartsCount = COUNT(*) FROM @MissingComponents;

-- 3. LOGIKA STATUSÓW I DATY
IF @v_MissingPartsCount > 0
   BEGIN
      -- SCENARIUSZ: BRAK CZĘŚCI (Status 4)
      SET @v_StatusID = 4; -- On Hold
      SELECT @v_MaxDelayDays = MAX(ShipTime) FROM @MissingComponents;
      SET @v_ScheduledDate = DATEADD(day, @v_MaxDelayDays + @v_ProductionTimeDays, GETDATE());
      PRINT 'Status: ON HOLD (Braki). Rezerwuję potrzebne ilości i czekam na dostawę.';
   END
ELSE
   BEGIN
      -- SCENARIUSZ: JEST OK (Status 2)
      SET @v_StatusID = 2;
      SET @v_ScheduledDate = DATEADD(day, @v_ProductionTimeDays, GETDATE());
      PRINT 'Status: IN PROGRESS. Pobieram materiały i startuję.';
   END

BEGIN TRANSACTION
   BEGIN TRY
      SELECT @v_NewProductionID = MAX(ProductionID) + 1 FROM Production;

      INSERT INTO Production (ProductionID, ProductID, EmployeeID, QuantityProduced,
                        StartDate, ScheduledDate, EndDate, Damaged, FineAssessed, ProductionPrice)
      VALUES (@v_NewProductionID, @p_ProductID, @p_EmployeeID, @p_QuantityToProduce,
            GETDATE(), @v_ScheduledDate, NULL, 0, NULL, @v_CalculatedPrice)

      IF @v_StatusID = 4
         BEGIN
            -- STATUS 4: Brakuje części -> Zwiększamy RESERVED QUANTITY
            -- Rezerwujemy tyle, ile wymaga całe zlecenie
            UPDATE c
            SET c.ReservedQuantity = c.ReservedQuantity +
                              (pd.ComponentQuantityToProduceProduct * @p_QuantityToProduce)
            FROM Component c
                  JOIN ProductDetails pd ON c.ComponentID = pd.ComponentID
            WHERE pd.ProductID = @p_ProductID;
         END
      ELSE
         IF @v_StatusID = 2
            BEGIN
               -- STATUS 2: Produkcja rusza -> Zmniejszamy QUANTITY IN STOCK
               UPDATE c
               SET c.QuantityInStock = c.QuantityInStock -
                                 (pd.ComponentQuantityToProduceProduct * @p_QuantityToProduce)
               FROM Component c
                     JOIN ProductDetails pd ON c.ComponentID = pd.ComponentID
               WHERE pd.ProductID = @p_ProductID;
            END

      -- C. Historia Statusów
```

```
        INSERT INTO StatusDetails (StatusID, ProductionID, StartDate, EndDate)
        VALUES (@v_StatusID, @v_NewProductionID, GETDATE(), NULL);

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        DECLARE @ErrMsg nvarchar(4000) = ERROR_MESSAGE();
        RAISERROR (@ErrMsg, 16, 1);
    END CATCH
    select @v_ScheduledDate as result

END;
go
```

6.11 **Koszt zamówienia** - zwraca koszt zamówienia

```
create procedure [dbo].[usp_GetOrderTotalCost]
        @p_ProductsList dbo.ListProductQuantity readonly
as
begin
  SET NOCOUNT ON;

  SELECT SUM(p.SellingPrice * pl.Quantity) AS TotalCost
  FROM Product p
  INNER JOIN @p_ProductsList pl ON p.ProductID = pl.ProductID;
end
```

6.12 **Przewidywana data zamówienia** - zwraca przewidywaną datę zamówienia

```
ALTER procedure [dbo].[usp_GetOrderDate]
        @p_ProductsList dbo.ListProductQuantity readonly
as
begin
        set nocount on;
        begin transaction;

        declare @v_ProductID int
        declare @v_MissingQuantity int
        declare @table_PlannedDate table (PlannedDate datetime);
        declare @v_PlannedDate datetime

        set @v_PlannedDate = GETDATE();
        declare @v_ResultPlannedDate datetime = GETDATE();

        declare cursor_missing_products cursor local fast_forward for
        select p.ProductID, abs(p.QuantityInStock - pl.Quantity) as MissingQuantity
        from Product p
        inner join @p_ProductsList pl
        on p.ProductID = pl.ProductID
        where p.QuantityInStock - pl.Quantity < 0

        open cursor_missing_products

        fetch next from cursor_missing_products
        into @v_ProductID, @v_MissingQuantity

        while @@FETCH_STATUS = 0
        begin
                delete from @table_PlannedDate;
```

```
                    SAVE TRANSACTION ProductionDate;
                        insert into @table_PlannedDate
                        exec dbo.usp_StartProduction
                            @p_ProductID = @v_ProductID,
                            @p_QuantityToProduce = @v_MissingQuantity,
                            @p_EmployeeID = 2;
                    ROLLBACK TRANSACTION ProductionDate;

                    select top 1 @v_ResultPlannedDate = PlannedDate from @table_PlannedDate;

                    if @v_ResultPlannedDate > @v_PlannedDate
    begin
       set @v_PlannedDate = @v_ResultPlannedDate;
    end

                    fetch next from cursor_missing_products into @v_ProductID, @v_MissingQuantity;
        end
        close cursor_missing_products;
        deallocate cursor_missing_products;

        SET @v_PlannedDate = DATEADD(day, 3, @v_PlannedDate);

        SELECT @v_PlannedDate AS OrderSheduledDate;

        commit transaction;
end
```

7. Triggery

7.1 Zabezpieczenie przed ujemnym stanem magazynowym komponentu

```
CREATE TRIGGER [dbo].[trg_Component_CheckNegativeStock]
ON [dbo].[Component]
AFTER UPDATE
AS
BEGIN
   SET NOCOUNT ON;

   IF EXISTS (SELECT 1 FROM inserted WHERE QuantityInStock < 0)
   BEGIN
      RAISERROR ('Błąd: Stan magazynowy komponentu nie może być ujemny.', 16, 1);
      ROLLBACK TRANSACTION;
   END
END;
```

7.2 Zabezpieczenie przed ujemnym stanem magazynowym produktu

```
CREATE TRIGGER [dbo].[trg_Product_CheckNegativeStock]
ON [dbo].[Product]
AFTER UPDATE
AS
BEGIN
   SET NOCOUNT ON;

   IF EXISTS (SELECT 1 FROM inserted WHERE QuantityInStock < 0)
   BEGIN
      RAISERROR ('Błąd: Stan magazynowy produktu nie może być ujemny.', 16, 1);
      ROLLBACK TRANSACTION;
   END
END;
```

7.3 Dodawanie produktu na produkcję gdy stan < MinStockLevel

```
CREATE TRIGGER [dbo].[trg_AutoStartProduction]
ON [dbo].[Product]
AFTER UPDATE
AS
BEGIN
   SET NOCOUNT ON;

   IF UPDATE(QuantityInStock)
   BEGIN
      DECLARE @ProdID int;

      DECLARE cur_LowStock CURSOR FOR
      SELECT ProductID
      FROM inserted
      WHERE QuantityInStock < MinStockLevel;

      OPEN cur_LowStock;

      FETCH NEXT FROM cur_LowStock INTO @ProdID;

      WHILE @@FETCH_STATUS = 0
      BEGIN

         EXEC dbo.usp_StartProduction
            @p_ProductID = @ProdID,
            @p_QuantityToProduce = 5,
            @p_EmployeeID = 3;

         FETCH NEXT FROM cur_LowStock INTO @ProdID;
      END

      CLOSE cur_LowStock;
      DEALLOCATE cur_LowStock;
   END
END;
```

7.4 Dodawanie zamówienia na komponent gdy stan magazynowy < MinStockLevel

```
CREATE TRIGGER [dbo].[trg_AutoOrderSupplies]
ON [dbo].[Component]
AFTER UPDATE
AS
BEGIN
   SET NOCOUNT ON;

   IF UPDATE(QuantityInStock)
   BEGIN
      DECLARE @CompID int;

      DECLARE cur_LowComponents CURSOR FOR
      SELECT ComponentID
      FROM inserted
      WHERE QuantityInStock < MinStockLevel;

      OPEN cur_LowComponents;

      FETCH NEXT FROM cur_LowComponents INTO @CompID;

      WHILE @@FETCH_STATUS = 0
      BEGIN
```

```
        EXEC dbo.usp_CreateSupplyOrder
            @p_ComponentID = @CompID,
            @p_Quantity = 50,
            @p_Discount = 0;

        FETCH NEXT FROM cur_LowComponents INTO @CompID;
    END

    CLOSE cur_LowComponents;
    DEALLOCATE cur_LowComponents;
  END
END;
```

8. Uprawnienia do danych

8.1 **Role_Sales** - Dział Handlowy

```
GRANT EXECUTE ON dbo.usp_CreateOrder TO Role_Sales;
GRANT EXECUTE ON dbo.usp_AddCompanyCustomer TO Role_Sales;
GRANT EXECUTE ON dbo.usp_AddPersonCustomer TO Role_Sales;
GRANT EXECUTE ON dbo.usp_CreateVat TO Role_Sales;

GRANT SELECT ON dbo.CurrentProductAvailableInStock TO Role_Sales;
GRANT SELECT ON dbo.ProductPriceAfterVat TO Role_Sales;
GRANT SELECT ON dbo.[All customers data] TO Role_Sales;
GRANT SELECT ON dbo.ProductCategorySale TO Role_Sales;
GRANT SELECT ON dbo.ShippersInfo TO Role_Sales;
```

8.2 **Role_Production** - Produckcja

```
GRANT EXECUTE ON dbo.usp_StartProduction TO Role_Production;
GRANT EXECUTE ON dbo.usp_UpdateProductionStatus TO Role_Production;
GRANT EXECUTE ON dbo.usp_AddComponent TO Role_Production;
GRANT EXECUTE ON dbo.usp_AddProduct TO Role_Production;
GRANT EXECUTE ON dbo.usp_CreateSupplyOrder TO Role_Production;
GRANT EXECUTE ON dbo.usp_CompleteSupplyOrder TO Role_Production;

GRANT SELECT ON dbo.[Missing Components] TO Role_Production;
GRANT SELECT ON dbo.[Missing Products] TO Role_Production;
GRANT SELECT ON dbo.ProductBillOfMaterials TO Role_Production;
GRANT SELECT ON dbo.ProductionPlanRaport TO Role_Production;
GRANT SELECT ON dbo.ProductionHistory TO Role_Production;
GRANT SELECT ON dbo.HalfProductsHistory TO Role_Production;
GRANT SELECT ON dbo.CurrentProduction TO Role_Production;
GRANT SELECT ON dbo.SuppliersInfo TO Role_Production;
```

8.3 **Role_Warehouse** - Magazyn

```
GRANT EXECUTE ON dbo.usp_CompleteSupplyOrder TO Role_Warehouse;
GRANT EXECUTE ON dbo.usp_UpdateProductionStatus TO Role_Warehouse;

GRANT SELECT ON dbo.CurrentProductAvailableInStock TO Role_Warehouse;
GRANT SELECT ON dbo.[Missing Components] TO Role_Warehouse;
```

8.4 **Role_Management** - Zarząd (tylko select'y)

```
GRANT SELECT ON dbo.MonthCategorySpend TO Role_Management;
GRANT SELECT ON dbo.MonthProductSpend TO Role_Management;
GRANT SELECT ON dbo.QuarterCategorySpends TO Role_Management;
GRANT SELECT ON dbo.QuarterProductSpends TO Role_Management;
GRANT SELECT ON dbo.YearCategorySpends TO Role_Management;
```

```
GRANT SELECT ON dbo.YearProductSpends TO Role_Management;
GRANT SELECT ON dbo.[Orders summary] TO Role_Management;
GRANT SELECT ON dbo.ProductionHistory TO Role_Management;
GRANT SELECT ON dbo.EmployeesInfo TO Role_Management;
GRANT SELECT ON dbo.EmployeeScheduleTemplate TO Role_Management;
```

8.5 **Role_Client** - zamawianie

```
GRANT EXECUTE ON dbo.usp_CreateOrder TO Role_Customer;
GRANT EXECUTE ON dbo.usp_GetOrderDate to Role_Customer;
GRANT EXECUTE ON dbo.usp_GetOrderTotalCost to Role_Customer;
GRANT SELECT ON dbo.CurrentProductAvailableInStock TO Role_Customer;
GRANT SELECT ON dbo.ProductPriceAfterVat TO Role_Customer;
GRANT SELECT ON dbo.ShippersInfo TO Role_Customer;
```

9.Indeksy

9.1 Filtrowanie zamówień po dacie

```
CREATE INDEX Orders_OrderDate
ON Orders (OrderDate)
INCLUDE (Freight, CustomerID);
```

9.2 Łączenie szczegółów zamówienia z produktami

```
CREATE INDEX OrderDetails_ProductID
ON OrderDetails (ProductID)
INCLUDE (Quantity, UnitPrice);
```

9.3 Grupowanie produktów po kategoriach

```
CREATE INDEX Product_ProductCategoryID
ON Product (ProductCategoryID)
INCLUDE (ProductName);
```

9.4 Filtrowanie produktów z małym stanem

```
CREATE INDEX Product_LowStock
ON Product (QuantityInStock)
WHERE QuantityInStock < 50;
```

9.5 Filtrowanie komponentów z małym stanem

```
CREATE INDEX Component_LowStock
ON Component (QuantityInStock)
WHERE QuantityInStock < 100;
```

9.6 Wyszukiwanie produkcji po dacie

```
CREATE INDEX Production_StatusDates
ON Production (ScheduledDate)
INCLUDE (ProductID, QuantityProduced);
```

9.7 Wyszukiwanie zamówień na komponenty po dacie

```
CREATE INDEX SupplyOrders_Dates
ON SupplyOrders (ScheduledShipDate, ShipmentDate);
```

9.8 Łączenie dostawców z komponentami

```
CREATE INDEX ComponentSupplyDetails_ComponentID
ON ComponentSupplyDetails (ComponentID);
```

9.9 Wyszukiwanie klienta po imieniu i nazwisku

```
CREATE INDEX CustomerPerson_Name
ON CustomerPerson (LastName, FirstName);
```