2017

APLIKACJA BAZODANOWA WSPOMAGAJĄCA PRACĘ HURTOWNI SPRZĘTU IT

BAZY DANYCH

KATARZYNA BŁASZKO-GRĄDZIK

Spis treści

Wstęp

Hurtownia sprzętu IT zajmuje się dystrybucją sprzętu i podzespołów informatycznych W trakcie bieżącej działalności zachodzi w jej wnętrzu cały szereg procesów o różnych priorytetach. Jednym z najbardziej charakterystycznych jest proces sprzedaży, opierający się o bezpośredni kontakt handlowy klient – sprzedawca. W swojej istocie jest on kluczowy dla działalności hurtowni, może stanowić w bezpośredni sposób o jej zyskach. Innym ważnym elementem jest zarządzanie obszarem magazynowym a co za tym idzie ilością posiadanego asortymentu. Wobec tak nakreślonych potrzeb, baza danych systemu informatycznego powinna realizować następujące zadania:

- gromadzenie danych klientów,
- gromadzenie danych dostawców i producentów,
- gromadzenie informacji o sprzęcie, jego cenie i dostępności w danej chwili,
- realizowanie transakcji sprzedaży, zamówień, transferu sprzętu.

Hurtownia sprzętu IT prowadzi swoją działalność w warunkach ostrej konkurencji rynkowej. W takiej sytuacji, każde działanie wspomagające funkcjonowanie przedsiębiorstwa wydaje się być jak najbardziej korzystne. Stworzenie wydajnej i dobrze spełniającej powyższe zadania aplikacji może w istotny sposób przyśpieszyć procesy wewnątrz firmy, a tym samym zapewnić jej sukces rynkowy.

1 Wykorzystane technologie

Niniejszy rozdział zawiera opis elementów, jakie zostały wykorzystane w trakcie projektowania systemu wspomagającego pracę hurtowni.

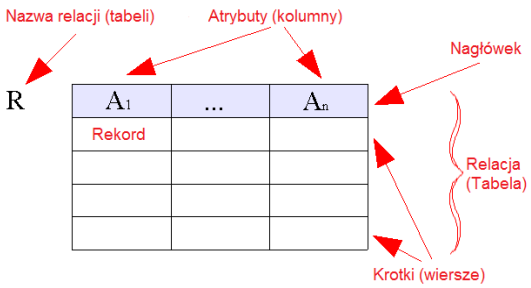
1.1 Relacyjne bazy danych

W najprostszym ujęciu relacyjną bazą danych jest baza złożona z przynajmniej dwóch tabel powiązanych ze sobą relacjami. Model relacyjnej bazy danych został stworzony przez Edgara Franka Codd'a, który to w swojej pracy w 1970 roku *A Relational Model of Data for Large Shared Data Banks*[1] opisał podstawowe zależności, jakie mogą występować pomiędzy danymi trwałymi, oraz wprowadził główne założenia dotyczące modelu. W swojej kolejnej pracy *Relational Completeness of Data Base Sublanguages* Codd uszczegółowił opis modelu oraz przedstawił dwa modele formalne przeszukiwania danych. W książce tej po raz pierwszy użył terminów algebry relacji oraz rachunku relacyjnego[2], pokazując, że oba modele są równoważne.

W roku 1979 firma Relational Software (obecnie Oracle) wypuściła na rynek pierwszy komercyjny relacyjny system zarządzania bazą danych (RDBMS ang. Relational Database Management Systems). Od tego momentu model relacyjny stał się dominującym podejściem do przechowywania trwałych danych zaś ilość badań i opracowań wokół tego tematu wzrosła lawinowo.

W dzisiejszym czasie funkcjonuje wiele spojrzeń na model relacyjny. Dwa główne podejścia to podejście formalne oraz podejście intuicyjne. Model ten jest powszechnie uważany za jeden z najważniejszych wynalazków w historii informatyki, dzięki któremu można elastycznie i oszczędnie operować danymi.

W modelu relacyjnym każda relacja (prezentowana w postaci np. tabeli) posiada unikalną nazwę, *nagłówek* i *zawartość*. Nagłówek relacji to zbiór atrybutów, gdzie atrybut jest parą *nazwa_atrybutu: nazwa_typu*, zawartość natomiast jest zbiorem krotek (reprezentowanych najczęściej w postaci wiersza w tabeli). W związku z tym, że nagłówek jest *zbiorem*



Rysunek 1 Schemat modelu relacyjnego
Źródło: www.wikipedia.pl

[1] E.F.Codd 'A Relational Model of Data for Large Shared Data Banks'
[2] E.F.Codd 'Relational Completeness of Data Base Sublanguages'

atrybutów nie jest ważna ich kolejność. Atrybuty zazwyczaj utożsamiane są z kolumnami tabeli. Każda krotka (wiersz) wyznacza zależność pomiędzy danymi w poszczególnych komórkach (np. osoba o danym numerze pesel posiada podane nazwisko i imię oraz adres)

Każda relacja (tabela) posiada tzw. *klucz główny* (ang. *primary key*)[3]. Klucz ten musi być unikatowy i może być kombinacją kilku kolumn, często jednak składa się tylko z jednej kolumny. Zadaniem klucza jest jednoznacznie identyfikować każdy wiersz.

Kolejnym rodzajem klucza jest tzw. klucz obcy (ang. *foreign key*). Służy on do wskazywania zależności między danymi zawartymi w różnych tabelach. Głównym zadaniem klucza w modelu relacyjnym jest sprawdzanie spójności danych w bazie, szczególnie dotyczy to kluczy obcych, na które nałożony jest wymóg, że w tabeli wskazywanej musi istnieć wartość klucza wskazującego.

Dodatkowym elementem modelu relacyjnego jest zbiór operacji służących do przeszukiwania i manipulacji danymi. Od strony formalnej takie zbiory operacji kojarzone są z tzw. *algebrą relacji* oraz z *rachunkiem relacyjnym*. Od strony praktycznej najbardziej popularnym językiem zapytań dla modelu relacyjnego jest język SQL

Przedstawienie relacji w postaci tabeli jest jedynie pewną reprezentacją graficzną, z punktu widzenia modelu relację można również przedstawić w postaci zbioru punktów w przestrzeni n-wymiarowej, gdzie punkt reprezentuje krotkę w relacji składającej się z n atrybutów.

1.2 Język SQL

Język SQL (ang. *Structured Query Language*) jest jednym z deklaratywnych języków zapytań używanych do tworzenia, modyfikowania, umieszczania oraz pobierania danych z baz danych. Został on opracowany w firmie IBM w latach 70-tych i stał się standardem w komunikacji z serwerami relacyjnych baz danych. Obecnie wiele systemów relacyjnych baz danych używa do komunikacji z użytkownikiem SQL, dlatego potocznie mówi się, że korzystanie z relacyjnych baz danych to korzystanie z SQL-a.

Pierwotną nazwą języka miał być *SEQUEL*, jednakże okazało się, że nazwa ta była już zastrzeżona przez brytyjską wytwórnię lotniczą Hawker Siddeley.

Język SQL nie posiada cech pozwalających na tworzenie kompletnych programów a tylko do komunikacji z bazą danych, dlatego też można powiedzieć, że

[3] *Encyclopedia of Database Systems*. Berlin: Springer US, 2009, ss. 2372-2375.

język ten jest podjęzykiem baz danych. Ze względu na jego wykorzystanie wyróżnia się 3 podstawowe formy SQL-a:

1. **interakcyjny** (autonomiczny) wykorzystywany jest przez użytkowników w celu bezpośredniego pobierania lub wprowadzania informacji do bazy.
2. **statyczny** kod SQL (Static SQL) pisany wraz z aplikacją, w której zostanie wykorzystany, jest pisany na początku i pozostaje niezmienny może jednak zawierać odwołania do zmiennych lub parametrów przekazujących wartości z lub do aplikacji.
3. **dynamiczny** kod SQL (Dynamic SQL) generowany jest w trakcie pracy aplikacji. Stosuje się go zamiast podejścia statycznego, jeżeli w chwili pisania aplikacji nie jest możliwe określenie zapytań, to znaczy, jeśli o treści zapytań decydować ma użytkownik.

Użycie SQL, zgodnie z jego nazwą, polega na zadawaniu zapytań do bazy danych. Zapytania można zaliczyć do jednego z trzech głównych podzbiorów:

- **SQL DML** (ang. *Data Manipulation Language* – „język manipulacji danymi") służy do umieszczania, kasowania, przeglądania oraz zmiany danych w bazie.
- **SQL DDL** (ang. *Data Definition Language* – „język definicji danych") służy do operacji na strukturach, czyli można dodawać, usuwać, zmieniać tabele lub bazy.

- **SQL DCL** (ang. *Data Control Language* – „język kontroli nad danymi"), służy do nadawania uprawnień dostępu do obiektów w bazie.

1.3 MS SQL Server

Microsoft SQL Server jest systemem zarządzania bazą danych, wspieranym i rozpowszechnianym przez korporację Microsoft, używa on języka zapytań Transact-SQL.

1.4 Język C#

Język C# jest obiektowym językiem programowania, zaprojektowanym przez firmy Microsoft do programowania platformy.NET Framework. Program pisany w tym języku podczas kompilacji jest przekształcany do języka CIL, czyli specjalnego kodu pośredniego wykonywanego w środowisku uruchomieniowym takim jak .NET

Framework, Mono lub DotGNU. Bez takiego środowiska wykonywanie tak skompilowanego programu nie jest możliwe. Język C# ma wiele cech wspólnych z takimi językami programowania jak Object Pascal, Delphi, C++ i Java.

2 Projekt systemu

Niniejszy rozdział zawiera opis projektowanego systemu.

2.1 Opis ogólny

Aplikacja będzie miała możliwość:

- przeglądania, dodawania, modyfikowania oraz usuwania danych zapisanych w bazie,
- generowania raportów,
- tworzenie kopi zapasowych,
- eksportowania danych do innych programów typu MS Excel.

Aplikacja będzie spełniała poniższe wymagania:

- prostota w użytkowaniu,
- system będzie współpracować z platformami Windows XP / Windows Vista / Windows 7,
- architektura klient-server.

2.2 Analiza wymagań

2.2.1 Wymagania funkcjonalne

Funkcje systemu sprzedażowego, korzystającego ze schematu:

- wprowadzanie oraz zarządzanie asortymentem hurtowni,
- rejestracja oraz zarządzanie personaliami klientów, dostawców oraz pracowników,
- realizacja transakcji sprzedaży,
- realizacja transakcji zakupu,
- możliwość dowolnego tworzenia zestawień sprzętowych na potrzeby klienta,
- realizacja dynamicznego zarządzania cenami asortymentu (dostosowanie oferty pod indywidualnego klienta).

2.2.2 Wymagania niefunkcjonalne

Wymagania związane z cechami produktu dotyczące:

- wydajności
- bezpieczeństwa
- kompatybilności z istniejącymi już aplikacjami w firmie

- użyteczności

2.3 Opisy modułów

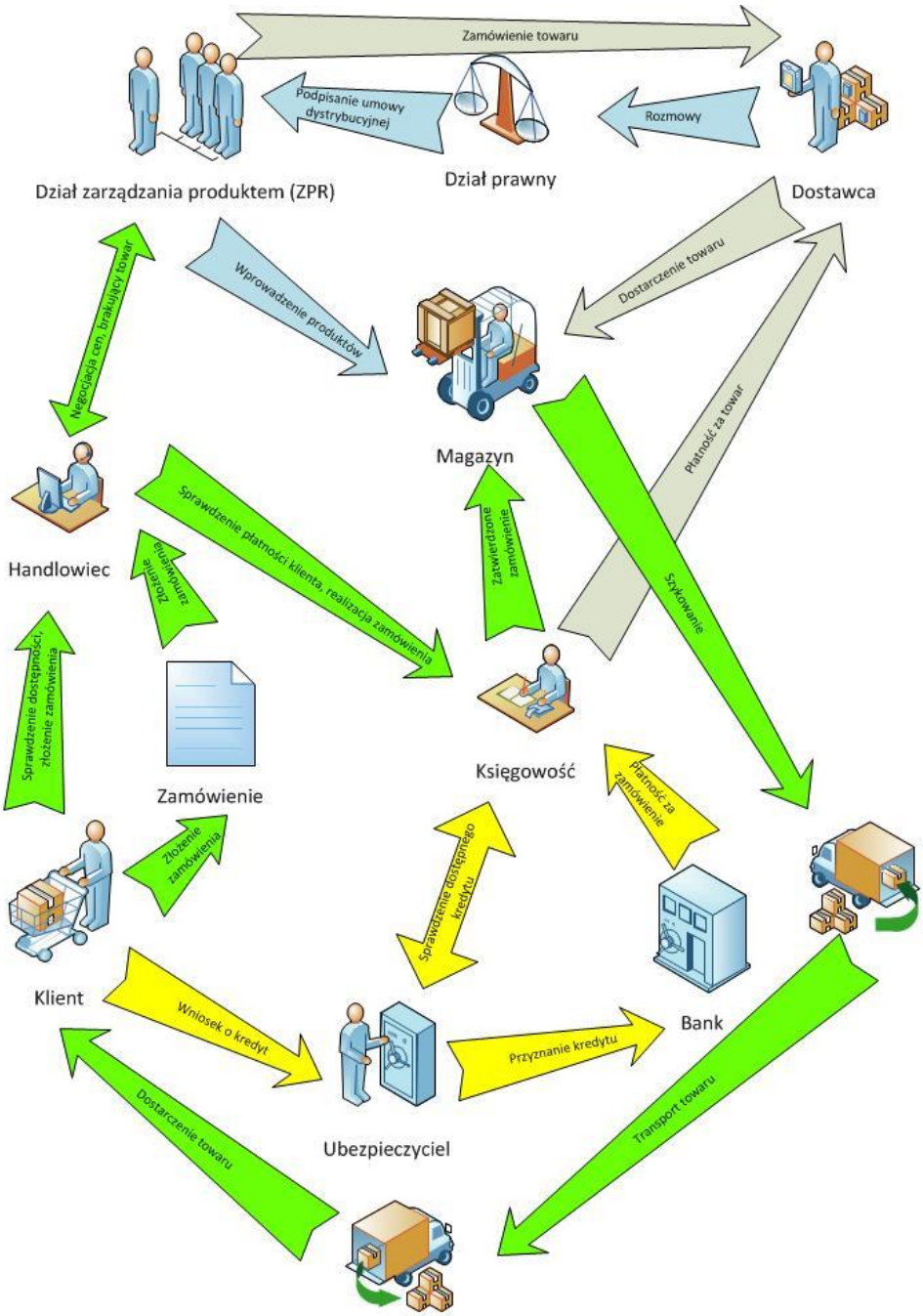
System będzie posiadał następujące moduły:

- Pracownicy informacja o pracownikach dostępna w formie ograniczonego odczytu dla każdego pracownika oraz w formie rozszerzonego odczytu i modyfikacji dla administratora i kadry zarządzającej. Celem modułu jest usprawnienie kontaktu między pracownikami danej firmy. Każdy pracownik może wyszukać innego sprawdzające jego adres e-mail, telefon oraz czym dany pracownik się zajmuje w firmie. Administratorzy mają dodatkowo możliwość modyfikacji lub dodania nowego pracownika.
- Towary informacja o towarach dostępna w formie odczytu dla każdego pracownika, oraz w formie odczytu i modyfikacji dla wyznaczonej grupy pracowników
- Kontrahenci informacja o kontrahentach dostępna w formie odczytu dla każdego pracownika, oraz w formie odczytu i modyfikacji dla wyznaczonej grupy pracowników
- Sprzedaż informacja o zleceniach sprzedaży i fakturach wystawianych kontrahentom oraz część realizująca sam proces zamówień i fakturowania.
- Magazyn informacja o stanach magazynowych, przyjęciach i zleceniach zakupów tworzonych przez product managerów.

2.4 Koncepcja działania i schemat logiczny

2.4.1 Diagram przepływu pracy

Diagram przepływu pracy przedstawia dynamiczne kształtowanie się poszczególnych działań wewnątrz przedsiębiorstwa. Daje on wgląd do powiązań między obiektami związanymi z pracą.



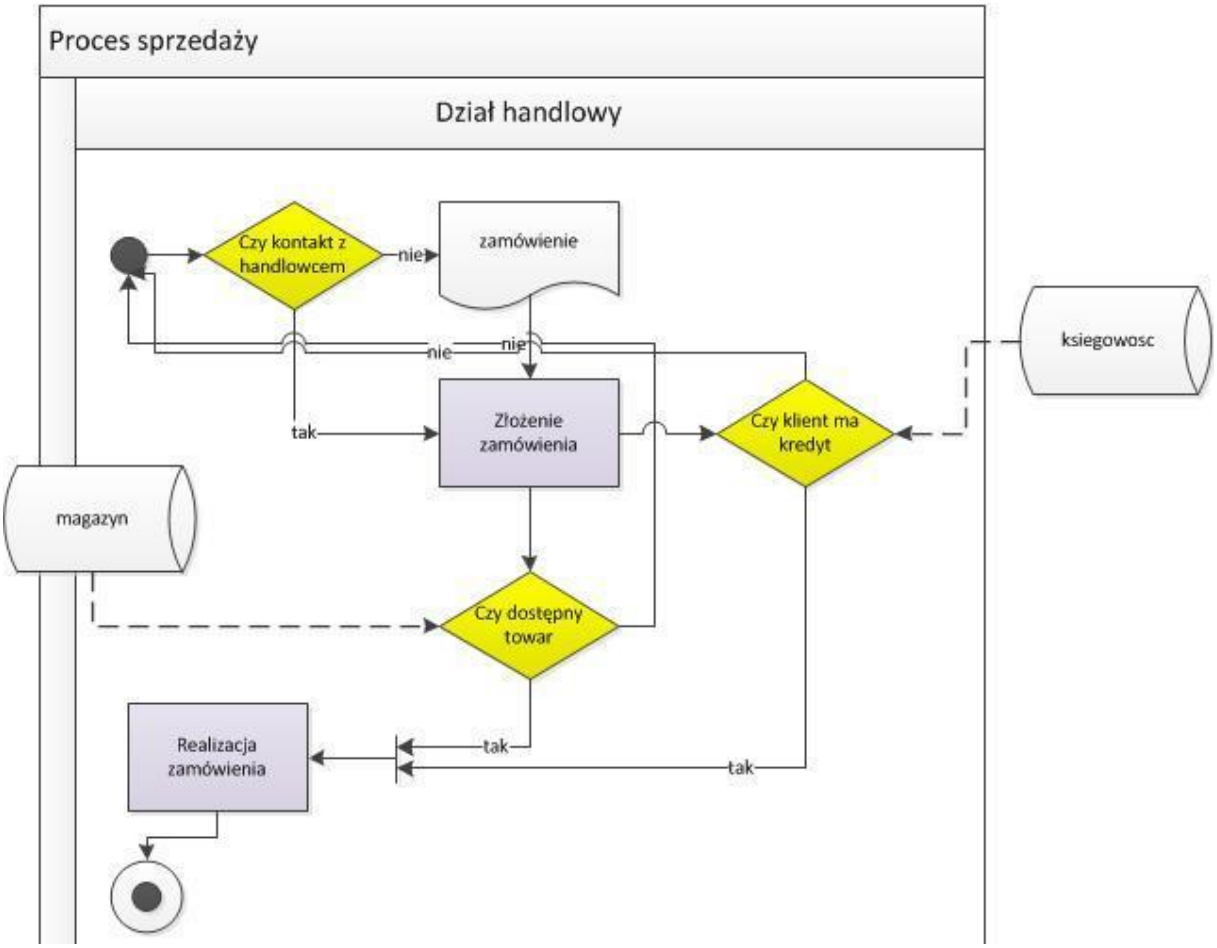
Rysunek 2 Diagram przepływu pracy w hurtowni sprzętu IT
Źródło: opracowanie własne

2.4.2 Diagramy procesów biznesowych

Diagramy procesów biznesowych przedstawiają podstawowe procesy zachodzące podczas działalności hurtowni sprzętu IT. Poniższe schematy obrazować będą kolejno:

- proces sprzedaży
- proces zamówień
- proces dodawania nowego dostawcy
- proces przyjmowania towaru

a) Proces sprzedaży



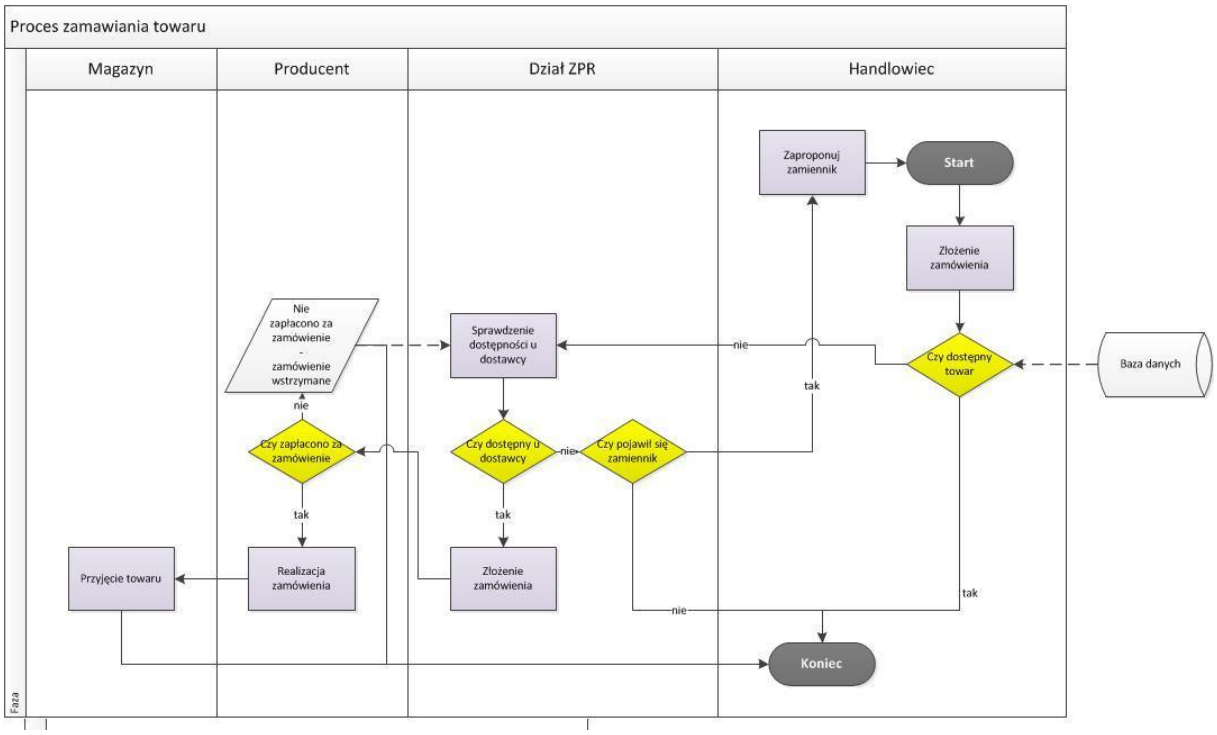
Rysunek 3 Diagram procesu biznesowego – sprzedaż
Źródło: opracowanie własne

Tabela 1 Opis procesów w procesie sprzedaży

Nazwa procesu	Opis
Złożenie zamówienia	Informacja otrzymana bezpośrednio przez system od klienta lub poprzez system o zawartości zamówienia.
Realizacja zamówienia	Przekazanie zamówienia do realizacji do magazynu

Źródło: opracowanie własne

b) Proces zamówień



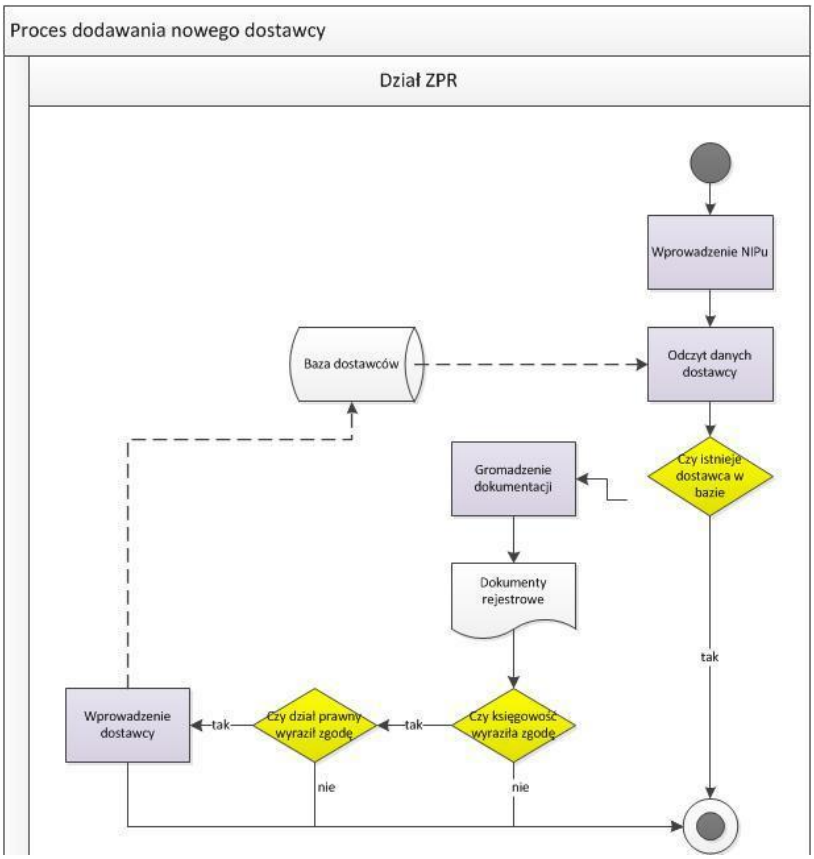
Rysunek 4 Diagram procesu biznesowego – zamówienia
Źródło: opracowanie własne

Tabela 2 Opis procesów w procesie zamówień

Nazwa procesu	Opis
Złożenie zamówienia (przez klienta)	Informacja otrzymana bezpośrednio przez system od klienta lub poprzez system o zawartości zamówienia.
Sprawdzenie dostępności u dostawcy	Sprawdzenie dostępności zamawianego przez klienta towaru u dostawcy
Zaproponuj zamiennik	Kontakt z klientem w sprawie wycofania towaru i zaproponowanie towaru zastępczego
Złożenie zamówienia (do dostawcy)	Złożenie zamówienia przez Product Managera do dostawcy
Realizacja zamówienia (przez dostawcę)	Wysyłka towaru na magazyn hurtowni
Przyjęcie towaru na magazyn	Fizyczne przyjęcie towaru na magazyn, przygotowanie do dalszej sprzedaży

Źródło: opracowanie własne

c) Proces dodawania nowego dostawcy



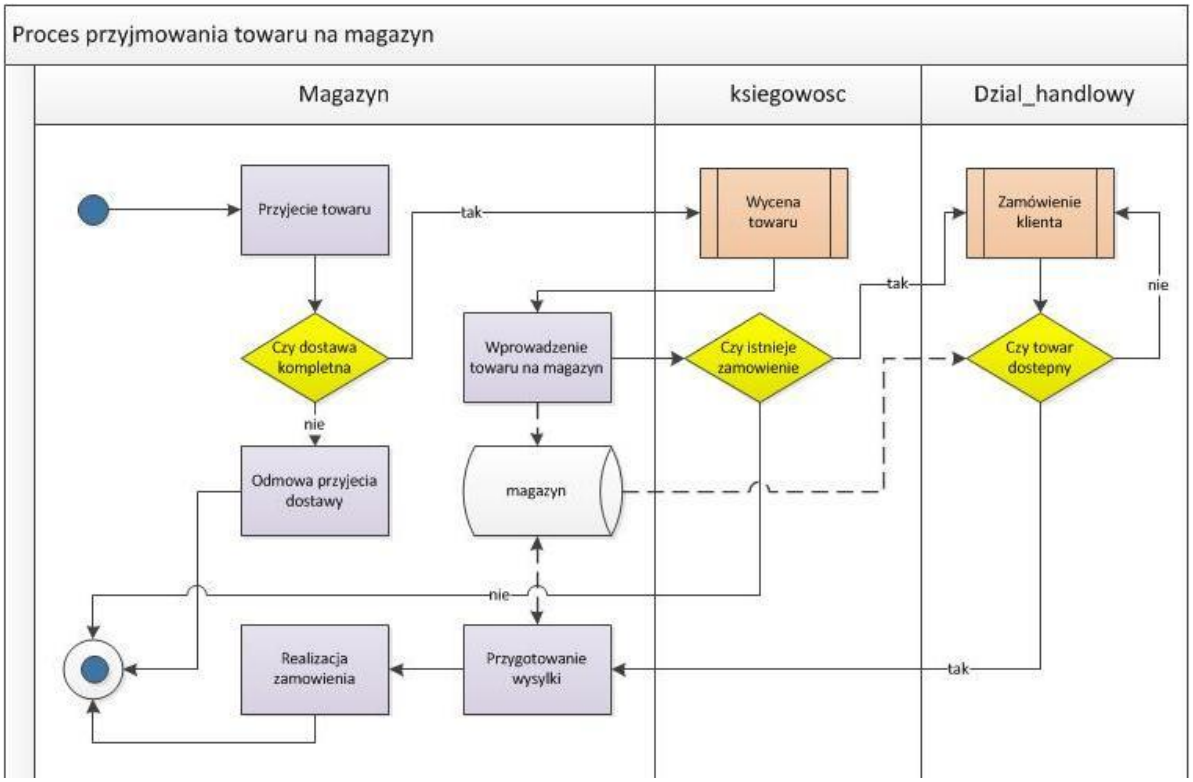
Rysunek 5 Diagram procesu biznesowego – dodawanie nowego dostawcy
Źródło: opracowanie własne

Tabela 3 Opis procesów w procesie dodawania nowego dostawcy

Nazwa procesu	Opis
Wprowadzenie NIPu	Informacja otrzymana bezpośrednio od klienta celem identyfikacji
Odczyt danych dostawcy	Sprawdzenie czy dostawca o podanym NIPie istnieje w bazie
Gromadzenie dokumentacji	Zbieranie dokumentacji rejestrowej od klienta
Wprowadzenie dostawcy	Wprowadzenie nowego dostawcy do bazy danych dostawców

Źródło: opracowanie własne

d) Proces przyjmowania towaru i realizacji zamówienia klienta



Rysunek 6 Diagram procesu biznesowego – przyjęcie towaru i realizacja zamówienia
Źródło: opracowanie własne

Tabela 4 Opis procesów w procesie przyjmowania towaru i realizacji zamówień

Nazwa procesu	Opis
Przyjęcie towaru	Przyjęcie towaru na dostawy w celu sprawdzenia kompletności dostawy
Odmowa przyjęcia dostawy	Niekompletna dostawa lub dostawa nieautoryzowana przez Product Managera – odesłanie towaru do dostawcy
Wprowadzenie towaru na magazyn	Wprowadzenie towaru na magazyn
Przygotowanie wysyłki	Przygotowanie towaru do wysłania do klienta, skompletowanie zamówienia
Realizacja zamówienia	Wysłanie towaru do klienta

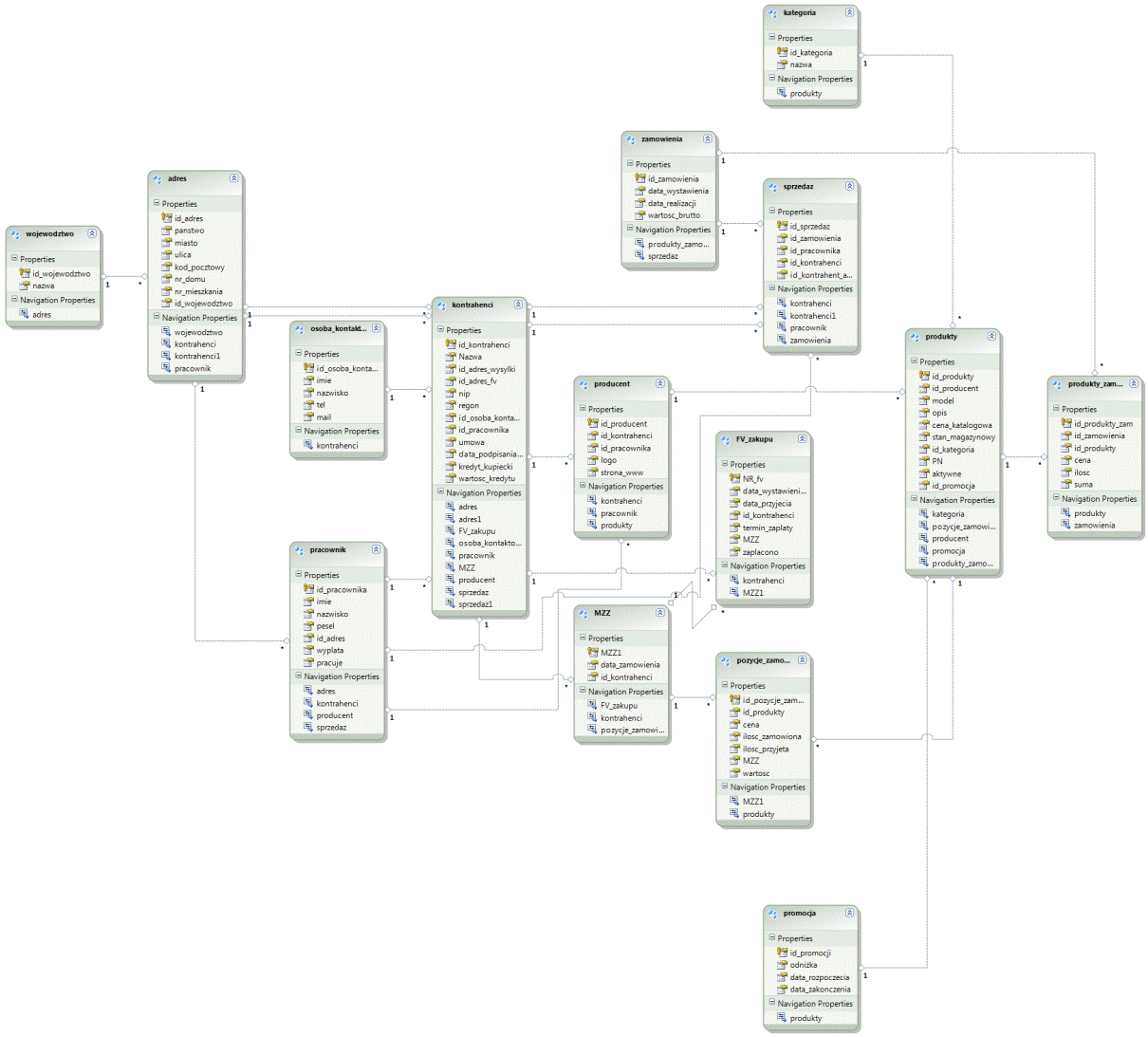
Źródło: opracowanie własne

3 Projekt bazy danych

Niniejszy rozdział przedstawia proces projektowania bazy danych oparty o model konceptualny oraz fizyczny.

3.1 Model konceptualny

Model CDM jest pomocny w ustaleniu głównych encji, ich atrybutów oraz relacji między nimi. Dokonuje także selekcji informacji, jaka będzie dostępna dla systemu. Jest to ogólny i abstrakcyjny model danych. Zwykle przedstawia go diagram ERD (Entity Relationship Diagram) zawierający oprócz atrybutów encji także same encje oraz strukturę powiązań między nimi wraz z liczebnością oraz kluczami głównymi i obcymi. Tworząc diagramy ERD zwykle przestrzega się pewnych reguł projektowych, m.in. zamieniając relacje „wiele do wielu” w relacje „jeden do wielu”.



Rysunek 7 Diagram ERD
Źródło: opracowanie własne

Szczegółowy opis utworzonych encji przedstawia poniższa tabela:

Tabela 5 Opis tabel i relacji w bazie

NAZWA ENCJI (TABELI)	OPIS (RELACJE)
ADRES	Jedna z podstawowych tabel systemu, która zawiera adresy kontrahentów, dostawców i pracowników Tabela tworzy relacje z poniższymi tabelami: 1. **WOJEWODZTWO (N:1)** 2. **KONTRAHENCI (1:N) – NA POLU ID_ADRES FV** 3. **KONTRAHENCI (1:N) – NA POLU ID_ADRES_WYSYLKI** Pola te mogą być mieć taką samą wartość, ale nie muszą 4. **PRACOWNIK (1:N)**
FV_ZAKUPU	Tabela zawiera informacje o zakupach dokonanych przez hurtownię, u kogo został zrobiony zakup, nr zlecenia zakupu z systemu, datę zakupu i datę przyjęcia towaru oraz informację czy faktura została zapłacona. Tabela tworzy relacje z poniższymi tabelami: 1. **KONTRAHENCI (N:1)** 2. **MZZ (N:1)** 3. **PRACOWNIK (1:N)**
POZYCJE_ZAMOWIENIA	Jest to tabela pomocnicza, umożliwiająca utworzenie relacji „wiele do wielu” pomiędzy tabelą **MZZ** i **PRODUKTY**
PRODUKTY_ZAMOWIENIA	Jest to tabela pomocnicza, umożliwiająca utworzenie relacji „wiele do wielu” pomiędzy tabelą **ZAMOWIENIA** i **PRODUKTY**
KONTRAHENCI	Podstawowa tabela w bazie danych, zawiera podstawowe informacje o kontrahentach hurtowni, który z pracowników odpowiada za kontakt z firmą, nazwę firmy, skan umowy oraz informację o przyznanym bądź nieprzyznanym kredycie kupieckim. Tabela tworzy relacje z następującymi tabelami: **1. ADRES (1:N)** **2. ADRES (1:N)** Tabela posiada podwójne odwołanie do tabeli adres ze względu na dodanie pola umożliwiającego wysłanie towaru na adres inny niż adres rejestrowy firmy. **3. OSOBA_KONTAKTOWA (1:N)** **4. PRACOWNIK (1:N)** **5. PRODUCENT (1:N)** **6. FV_ZAKUPU (1:N)** **7. MZZ (1:N)** **8. SPRZEDAZ (1:N)**
MZZ	Meta zlecenie zakupu, tabela zawierająca informacje o zamówieniach, kto, kiedy i gdzie dokonał zakupów. FV zakupu musi zawierać nr mzz.

OSOBA_KONTAKTOWA Tabela zawierająca informację o osobach w firmie kontrahenta wyznaczoną do kontaktu z hurtownią, klient może mieć zgłoszonych kilka osób o różnym poziomie dostępu. Wiąże się bezpośrednio z tabelą **KONTRAHENCI (N:1).**

PRACOWNIK Tabela zawierająca informację o pracownikach, **ID_PRACOWNIKA** tworzone jest na podstawie działu, w którym pracuje oraz unikalnego trzycyfrowego numeru.

Działy:

- **LGM** – logistyka i magazyn
- **ZPR** – dział zarządzania produktem
- **SPR** – dział sprzedaży
- **FIK** – dział finansów i księgowości

Tworzy relację z tabelami:

1. **MZZ (1:N) –** informacja, kto zamówił towar u dostawcy
2. **PRODUCENT (1:N)** – informacja, kto jest odpowiedzialny za kontakt z danym dostawcą
3. **SPRZEDAZ (1:N)** – kto wystawił fv sprzedaży
4. **KONTRAHENCI (1:N)** – kto obsługuje danego **KLIENTA**
5. **ADRES (N:1)**

Tabela posiada również pole **PRACUJE, KTÓRE** ma za zadanie określić czy dany pracownik jeszcze pracuje w danej firmie.

PRODUCENT Tabela zawierające dane producentów, których sprzęt jest sprzedawany przez hurtownię.

PRODUKTY Jedna z podstawowych tabel systemu, która zawiera informacje o produktach sprzedawanych przez hurtownię.

Tabela tworzy relacje z poniższymi tabelami:

6. **KATEGORIA (N:1)**
7. **PROMOCJA (N:1)**
8. **POZYCJE_ZAMOWIENIA (1:N)**
9. **PRODUCENT (N:1)**
10. **PRODUKTY_ZAMOWIENIA (1:N)**

PROMOCJA Tabela pozwalająca nie ingerując w stałą cenę na ustawienie promocji cenowej w wybranym okresie czasu. W systemie po upływie okresu promocji znów będzie wyświetlana cena podstawowa. Odnosi się tylko do tabeli **PRODUKTY**.

SPRZEDAZ Jedna z podstawowych tabel systemu, zawiera informacje o sprzedaży produktów do klientów. Jest ostateczną formą sprzedaży oznaczającą, że towar został wydany klientowi.

Tabela tworzy relacje z tabelami:

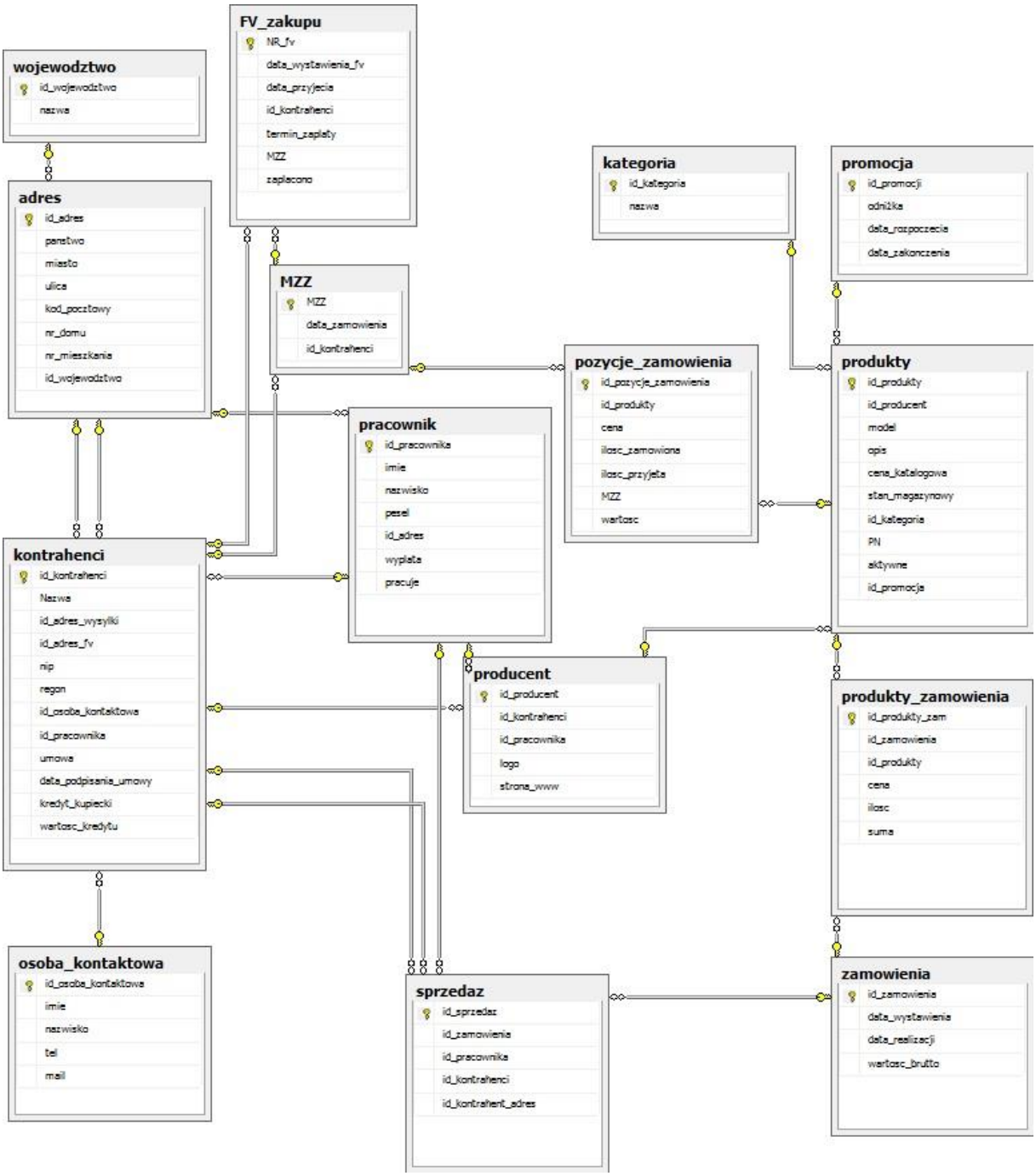
1. **KONTRAHENCI (1:N)**
2. **PRACOWNIK (1:N)**
3. **ZAMOWIENIA (1:N)**

WOJEWÓDZTWO	Tabela słownikowa zawierająca informację o wszystkich województwach.
ZAMOWIENIA	Tabela zawierające informacje o zamówieniach klienta, przedstawiane klientowi przed wystawieniem faktury do zatwierdzenia. Tworzy relacje: 1. **SPRZEDAZ (1:N)** 2. **PRODUKTY_ZAMOWIENIA (1:N)**
KATEGORIA	Tabela słownikowa zawierająca informacje o kategoriach, na jakie podzielone są produkty w hurtowni.

Źródło: opracowanie własne

3.2 Fizyczny model bazy danych

Model fizyczny – PDM jest ściśle związany z konkretnym silnikiem baz danych. Poniższy model został pozbawiony atrybutów ze względu na większą czytelność.



Rysunek 8 Fizyczny model bazy danych
Źródło: opracowanie własne

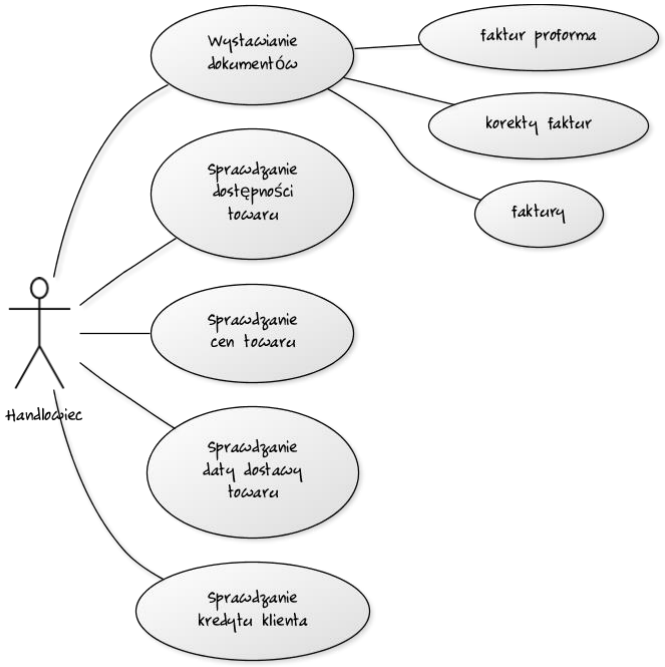
3.3 Bezpieczeństwo i dostęp

3.3.1 Role i diagramy przypadków użycia

Diagramy przypadków użycia powstają głównie w celu modelowania zakresu funkcjonalnego systemu oraz jako źródło przypadków testowych opisując w sposób graficzny uprawnienia ról występujących w systemie.

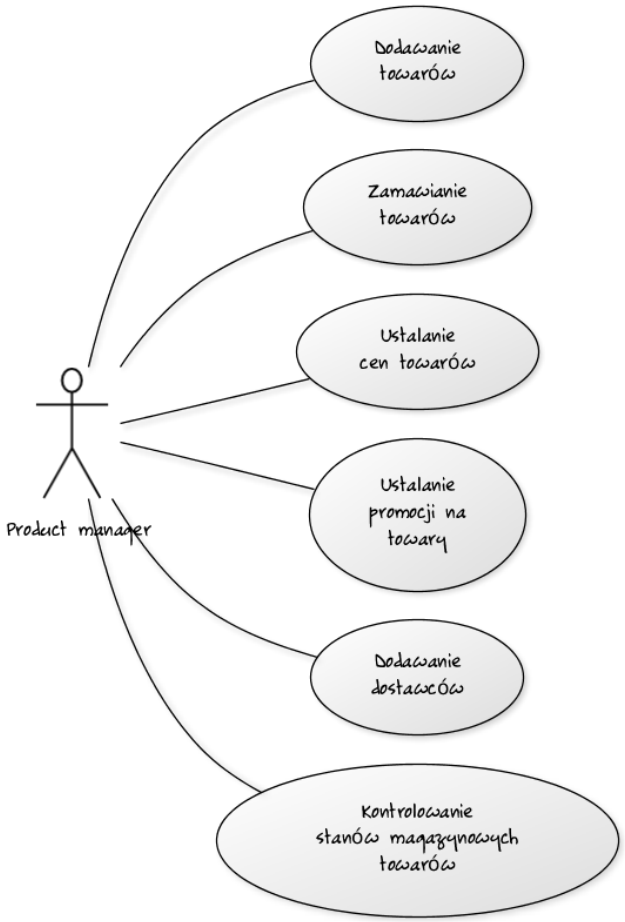
W systemie wyróżnione zostały następujące role:

- handlowiec – dostęp do aplikacji poprzez interfejs aplikacji z uprawnieniami do przeglądania, dodawania i modyfikowania istniejących danych dotyczących klientów oraz z uprawnieniami do przeglądania danych produktów,



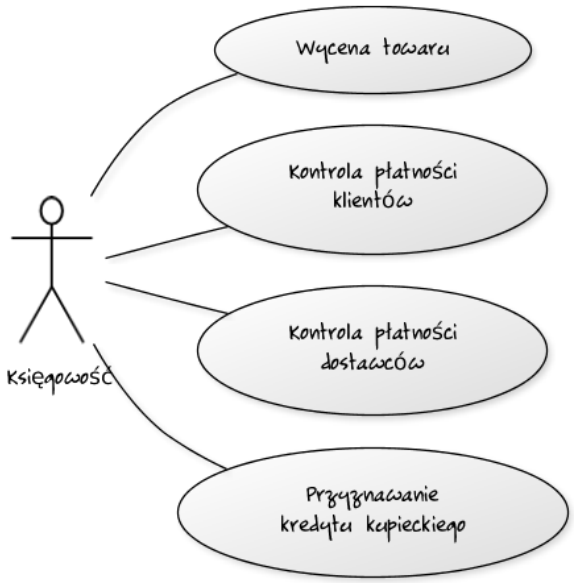
Rysunek 9 Diagram przypadku użycia – Handlowiec
Źródło: opracowanie własne

- product manager – dostęp do aplikacji poprzez interfejs aplikacji z uprawnieniami do przeglądania, dodawania i modyfikowania produktów, zleceń zakupów, ustalania i modyfikowania cen oraz promocji,



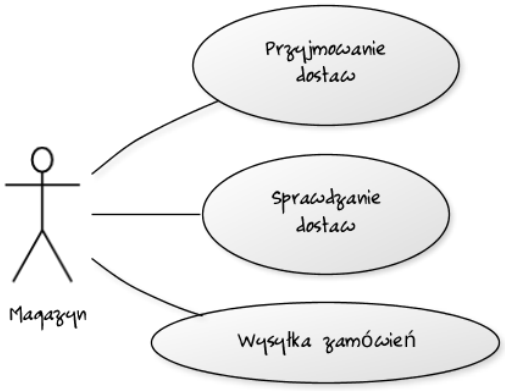
Rysunek 10 Diagram przypadku użycia - Product Manager
Źródło: opracowanie własne

- księgowość - dostęp do aplikacji poprzez interfejs aplikacji z uprawnieniami do przeglądania, modyfikowania danych klientów dotyczących kredytów,



Rysunek 11 Diagram przypadku użycia – Księgowość
Źródło: opracowanie własne

- magazynier - dostęp do aplikacji poprzez interfejs aplikacji z uprawnieniami do przeglądania, dodawania i modyfikowania istniejących zleceń zakupów.



Rysunek 12 Diagram przypadku użycia – Magazyn
Źródło: opracowanie własne

- administrator – dostęp do wszystkich komponentów systemu,
- zarząd - dostęp do wszystkich komponentów systemu.

3.3.2 Tabela uprawnień

Tabela uprawnień określa uprawnienia użytkownika w stosunku do tabel:

- O – odczyt
- M – modyfikacja
- D – dodawanie
- U – usuwanie
- „-„ – brak dostępu
- „+” – pełny dostęp.

Poniżej przedstawiono tabelę uprawnień dla projektowanego systemu.

Tabela 6 Tabela uprawnień

rola	Tabela														
	ADRES	**FV_ZAKUPU**	**POZYCJE_ZAMOWIENIA**	**PRODUKTY_ZAMOWIENIA**	**KONTRAHENCI**	**MZZ**	**OSOBA_KONTAKTOWA**	**PRACOWNIK**	**PRODUCENT**	**PRODUKTY**	**PROMOCJA**	**SPRZEDAZ**	**WOJEWÓDZTWO**	**ZAMOWIENIA**	**KATEGORIA**
Administrator/zarząd	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Magazynier	O	O	OM	O	O	OM	-	O	O	O	-	-	O	O	O
Księgowość	O	O	-	-	O	OM	O	OM	O	O	O	O	O	O	O
Handlowiec	OMD	OD	O	OMDU	OMD	O	OMD	O	O	O	O	OMD	O	OMD	O
Product Manager	OMD	O	OMD	O	OMD	OMD	OMD	O	OMD	OMD	OMD	O	O	O	O

Źródło: opracowanie własne

3.4 Realizacja bazy danych

Poniżej znajduje się skrypt SQL bazy danych dla projektowanego systemu, wygenerowany za pomocą MS SQL Server Managment Studio.

```
USE [hurtownia]
GO
/****** Object:  Table [dbo].[promocja]    Script Date: 08/01/2011
15:43:59 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[promocja](
	[id_promocji] [int] IDENTITY(1,1) NOT NULL,
	[obnizka] [decimal](19, 4) NOT NULL,
	[data_rozpoczecia] [datetime] NOT NULL,
	[data_zakonczenia] [datetime] NOT NULL,
 CONSTRAINT [PK_promocja] PRIMARY KEY CLUSTERED
(
	[id_promocji] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[zamowienia]    Script Date: 08/01/2011
15:43:59 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[zamowienia](
	[id_zamowienia] [int] IDENTITY(1,1) NOT NULL,
	[data_wystawienia] [datetime] NOT NULL,
	[data_realizacji] [datetime] NOT NULL,
	[wartosc_brutto] [decimal](19, 4) NOT NULL,
 CONSTRAINT [PK_zamowienia] PRIMARY KEY CLUSTERED
(
	[id_zamowienia] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[wojewodztwo]    Script Date: 08/01/2011
15:43:59 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[wojewodztwo](
	[id_wojewodztwo] [int] IDENTITY(1,1) NOT NULL,
	[nazwa] [nchar](30) NOT NULL,
 CONSTRAINT [PK_wojewodztwo] PRIMARY KEY CLUSTERED
(
	[id_wojewodztwo] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

```
/****** Object:  Table [dbo].[osoba_kontaktowa]    Script Date:
08/01/2011 15:43:59 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[osoba_kontaktowa](
	[id_osoba_kontaktowa] [int] IDENTITY(1,1) NOT NULL,
	[imie] [nvarchar](30) NOT NULL,
	[nazwisko] [nvarchar](30) NOT NULL,
	[tel] [nvarchar](15) NOT NULL,
	[mail] [nvarchar](max) NOT NULL,
 CONSTRAINT [PK_osoba_kontaktowa] PRIMARY KEY CLUSTERED
(
	[id_osoba_kontaktowa] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[kategoria]    Script Date: 08/01/2011
15:43:59 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[kategoria](
	[id_kategoria] [int] IDENTITY(1,1) NOT NULL,
	[nazwa] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_kategoria] PRIMARY KEY CLUSTERED
(
	[id_kategoria] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[adres]    Script Date: 08/01/2011
15:43:59 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[adres](
	[id_adres] [int] IDENTITY(1,1) NOT NULL,
	[panstwo] [nchar](20) NOT NULL,
	[miasto] [nchar](20) NOT NULL,
	[ulica] [nchar](30) NOT NULL,
	[kod_pocztowy] [nchar](6) NOT NULL,
	[nr_domu] [nchar](10) NOT NULL,
	[nr_mieszkania] [nchar](10) NOT NULL,
	[id_wojewodztwo] [int] NOT NULL,
 CONSTRAINT [PK_adres] PRIMARY KEY CLUSTERED
(
	[id_adres] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[pracownik]    Script Date: 08/01/2011
15:43:59 ******/
SET ANSI_NULLS ON
GO
```

```
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[pracownik](
	[id_pracownika] [nvarchar](7) NOT NULL,
	[imie] [nchar](30) NOT NULL,
	[nazwisko] [nchar](30) NOT NULL,
	[pesel] [char](11) NOT NULL,
	[id_adres] [int] NOT NULL,
	[wyplata] [money] NOT NULL,
	[pracuje] [binary](1) NOT NULL,
 CONSTRAINT [PK_pracownik] PRIMARY KEY CLUSTERED
(
	[id_pracownika] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[kontrahenci]    Script Date: 08/01/2011
15:43:59 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[kontrahenci](
	[id_kontrahenci] [int] NOT NULL,
	[Nazwa] [nvarchar](30) NOT NULL,
	[id_adres_wysylki] [int] NOT NULL,
	[id_adres_fv] [int] NOT NULL,
	[nip] [nvarchar](10) NOT NULL,
	[regon] [int] NOT NULL,
	[id_osoba_kontaktowa] [int] NOT NULL,
	[id_pracownika] [nvarchar](7) NOT NULL,
	[umowa] [bit] NOT NULL,
	[data_podpisania_umowy] [datetime] NOT NULL,
	[kredyt_kupiecki] [binary](1) NOT NULL,
	[wartosc_kredytu] [money] NULL,
 CONSTRAINT [PK_kontrahenci] PRIMARY KEY CLUSTERED
(
	[id_kontrahenci] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[sprzedaz]    Script Date: 08/01/2011
15:43:59 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[sprzedaz](
	[id_sprzedaz] [int] IDENTITY(1,1) NOT NULL,
	[id_zamowienia] [int] NOT NULL,
	[id_pracownika] [nvarchar](7) NOT NULL,
```

```
	[id_kontrahenci] [int] NOT NULL,
 CONSTRAINT [PK_sprzedaz_] PRIMARY KEY CLUSTERED
(
	[id_sprzedaz] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[producent]    Script Date: 08/01/2011
15:43:59 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[producent](
	[id_producent] [int] IDENTITY(1,1) NOT NULL,
	[id_kontrahenci] [int] NOT NULL,
	[id_pracownika] [nvarchar](7) NOT NULL,
	[logo] [bit] NOT NULL,
	[strona_www] [nvarchar](max) NOT NULL,
 CONSTRAINT [PK_producent] PRIMARY KEY CLUSTERED
(
	[id_producent] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[MZZ]    Script Date: 08/01/2011 15:43:59
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[MZZ](
	[MZZ] [int] NOT NULL,
	[data_zamowienia] [date] NOT NULL,
	[id_pracownika] [nvarchar](7) NOT NULL,
	[id_kontrahenci] [int] NOT NULL,
 CONSTRAINT [PK_MZZ] PRIMARY KEY CLUSTERED
(
	[MZZ] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[FV_zakupu]    Script Date: 08/01/2011
15:43:59 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[FV_zakupu](
	[NR_fv] [nvarchar](15) NOT NULL,
	[data_wystawienia_fv] [date] NOT NULL,
	[data_przyjecia] [date] NOT NULL,
	[id_kontrahenci] [int] NOT NULL,
	[termin_zaplaty] [int] NOT NULL,
	[MZZ] [int] NOT NULL,
	[zaplacono] [binary](1) NOT NULL,
```

```
 CONSTRAINT [PK_FV_zakupu] PRIMARY KEY CLUSTERED
(
	[NR_fv] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[produkty]    Script Date: 08/01/2011
15:43:59 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[produkty](
	[id_produkty] [int] IDENTITY(1,1) NOT NULL,
	[id_producent] [int] NOT NULL,
	[model] [nchar](30) NOT NULL,
	[opis] [varchar](max) NULL,
	[cena_katalogowa] [decimal](19, 4) NOT NULL,
	[stan_magazynowy] [int] NOT NULL,
	[id_kategoria] [int] NOT NULL,
	[PN] [nvarchar](20) NOT NULL,
	[aktywne] [binary](1) NOT NULL,
	[id_promocja] [int] NOT NULL,
 CONSTRAINT [PK_produkty] PRIMARY KEY CLUSTERED
(
	[id_produkty] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[produkty_zamowienia]    Script Date:
08/01/2011 15:43:59 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[produkty_zamowienia](
	[id_produkty_zam] [int] IDENTITY(1,1) NOT NULL,
	[id_zamowienia] [int] NOT NULL,
	[id_produkty] [int] NOT NULL,
	[cena] [money] NOT NULL,
	[ilosc] [int] NOT NULL,
	[suma] [money] NOT NULL,
 CONSTRAINT [PK_produkty_zamowienia] PRIMARY KEY CLUSTERED
(
	[id_produkty_zam] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[pozycje_zamowienia]    Script Date:
08/01/2011 15:43:59 ******/
SET ANSI_NULLS ON
GO
```

```
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[pozycje_zamowienia](
	[id_pozycje_zamowienia] [int] NOT NULL,
	[id_produkty] [int] NOT NULL,
	[cena] [money] NULL,
	[ilosc_zamowiona] [int] NOT NULL,
	[ilosc_przyjeta] [int] NULL,
	[MZZ] [int] NOT NULL,
	[wartosc] [money] NULL,
 CONSTRAINT [PK_pozycje_zamowienia] PRIMARY KEY CLUSTERED
(
	[id_pozycje_zamowienia] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  ForeignKey [FK_adres_wojewodztwo]    Script Date:
08/01/2011 15:43:59 ******/
ALTER TABLE [dbo].[adres]  WITH CHECK ADD  CONSTRAINT
[FK_adres_wojewodztwo] FOREIGN KEY([id_wojewodztwo])
REFERENCES [dbo].[wojewodztwo] ([id_wojewodztwo])
GO
ALTER TABLE [dbo].[adres] CHECK CONSTRAINT [FK_adres_wojewodztwo]
GO
/****** Object:  ForeignKey [FK_FV_zakupu_kontrahenci]    Script Date:
08/01/2011 15:43:59 ******/
ALTER TABLE [dbo].[FV_zakupu]  WITH CHECK ADD  CONSTRAINT
[FK_FV_zakupu_kontrahenci] FOREIGN KEY([id_kontrahenci])
REFERENCES [dbo].[kontrahenci] ([id_kontrahenci])
GO
ALTER TABLE [dbo].[FV_zakupu] CHECK CONSTRAINT
[FK_FV_zakupu_kontrahenci]
GO
/****** Object:  ForeignKey [FK_FV_zakupu_MZZ]    Script Date:
08/01/2011 15:43:59 ******/
ALTER TABLE [dbo].[FV_zakupu]  WITH CHECK ADD  CONSTRAINT
[FK_FV_zakupu_MZZ] FOREIGN KEY([MZZ])
REFERENCES [dbo].[MZZ] ([MZZ])
GO
ALTER TABLE [dbo].[FV_zakupu] CHECK CONSTRAINT [FK_FV_zakupu_MZZ]
GO
/****** Object:  ForeignKey [FK_kontrahenci_adres]    Script Date:
08/01/2011 15:43:59 ******/
ALTER TABLE [dbo].[kontrahenci]  WITH CHECK ADD  CONSTRAINT
[FK_kontrahenci_adres] FOREIGN KEY([id_adres_wysylki])
REFERENCES [dbo].[adres] ([id_adres])
GO
ALTER TABLE [dbo].[kontrahenci] CHECK CONSTRAINT
[FK_kontrahenci_adres]
GO
/****** Object:  ForeignKey [FK_kontrahenci_adres1]    Script Date:
08/01/2011 15:43:59 ******/
ALTER TABLE [dbo].[kontrahenci]  WITH CHECK ADD  CONSTRAINT
[FK_kontrahenci_adres1] FOREIGN KEY([id_adres_fv])
REFERENCES [dbo].[adres] ([id_adres])
GO
ALTER TABLE [dbo].[kontrahenci] CHECK CONSTRAINT
[FK_kontrahenci_adres1]
GO
```

```
/****** Object:  ForeignKey [FK_kontrahenci_osoba_kontaktowa]
Script Date: 08/01/2011 15:43:59 ******/
ALTER TABLE [dbo].[kontrahenci]  WITH CHECK ADD  CONSTRAINT
[FK_kontrahenci_osoba_kontaktowa] FOREIGN KEY([id_osoba_kontaktowa])
REFERENCES [dbo].[osoba_kontaktowa] ([id_osoba_kontaktowa])
GO
ALTER TABLE [dbo].[kontrahenci] CHECK CONSTRAINT
[FK_kontrahenci_osoba_kontaktowa]
GO
/****** Object:  ForeignKey [FK_kontrahenci_pracownik]    Script Date:
08/01/2011 15:43:59 ******/
ALTER TABLE [dbo].[kontrahenci]  WITH CHECK ADD  CONSTRAINT
[FK_kontrahenci_pracownik] FOREIGN KEY([id_pracownika])
REFERENCES [dbo].[pracownik] ([id_pracownika])
GO
ALTER TABLE [dbo].[kontrahenci] CHECK CONSTRAINT
[FK_kontrahenci_pracownik]
GO
/****** Object:  ForeignKey [FK_MZZ_kontrahenci]    Script Date:
08/01/2011 15:43:59 ******/
ALTER TABLE [dbo].[MZZ]  WITH CHECK ADD  CONSTRAINT
[FK_MZZ_kontrahenci] FOREIGN KEY([id_kontrahenci])
REFERENCES [dbo].[kontrahenci] ([id_kontrahenci])
GO
ALTER TABLE [dbo].[MZZ] CHECK CONSTRAINT [FK_MZZ_kontrahenci]
GO
/****** Object:  ForeignKey [FK_MZZ_pracownik]    Script Date:
08/01/2011 15:43:59 ******/
ALTER TABLE [dbo].[MZZ]  WITH CHECK ADD  CONSTRAINT [FK_MZZ_pracownik]
FOREIGN KEY([id_pracownika])
REFERENCES [dbo].[pracownik] ([id_pracownika])
GO
ALTER TABLE [dbo].[MZZ] CHECK CONSTRAINT [FK_MZZ_pracownik]
GO
/****** Object:  ForeignKey [FK_pozycje_zamowienia_MZZ]    Script
Date: 08/01/2011 15:43:59 ******/
ALTER TABLE [dbo].[pozycje_zamowienia]  WITH CHECK ADD  CONSTRAINT
[FK_pozycje_zamowienia_MZZ] FOREIGN KEY([MZZ])
REFERENCES [dbo].[MZZ] ([MZZ])
GO
ALTER TABLE [dbo].[pozycje_zamowienia] CHECK CONSTRAINT
[FK_pozycje_zamowienia_MZZ]
GO
/****** Object:  ForeignKey [FK_pozycje_zamowienia_produkty]    Script
Date: 08/01/2011 15:43:59 ******/
ALTER TABLE [dbo].[pozycje_zamowienia]  WITH CHECK ADD  CONSTRAINT
[FK_pozycje_zamowienia_produkty] FOREIGN KEY([id_produkty])
REFERENCES [dbo].[produkty] ([id_produkty])
GO
ALTER TABLE [dbo].[pozycje_zamowienia] CHECK CONSTRAINT
[FK_pozycje_zamowienia_produkty]
GO
/****** Object:  ForeignKey [FK_pracownik_adres]    Script Date:
08/01/2011 15:43:59 ******/
ALTER TABLE [dbo].[pracownik]  WITH CHECK ADD  CONSTRAINT
[FK_pracownik_adres] FOREIGN KEY([id_adres])
REFERENCES [dbo].[adres] ([id_adres])
GO
ALTER TABLE [dbo].[pracownik] CHECK CONSTRAINT [FK_pracownik_adres]
GO
```

```
/****** Object:  ForeignKey [FK_producent_kontrahenci]    Script Date:
08/01/2011 15:43:59 ******/
ALTER TABLE [dbo].[producent]  WITH CHECK ADD  CONSTRAINT
[FK_producent_kontrahenci] FOREIGN KEY([id_kontrahenci])
REFERENCES [dbo].[kontrahenci] ([id_kontrahenci])
GO
ALTER TABLE [dbo].[producent] CHECK CONSTRAINT
[FK_producent_kontrahenci]
GO
/****** Object:  ForeignKey [FK_producent_pracownik]    Script Date:
08/01/2011 15:43:59 ******/
ALTER TABLE [dbo].[producent]  WITH CHECK ADD  CONSTRAINT
[FK_producent_pracownik] FOREIGN KEY([id_pracownika])
REFERENCES [dbo].[pracownik] ([id_pracownika])
GO
ALTER TABLE [dbo].[producent] CHECK CONSTRAINT
[FK_producent_pracownik]
GO
/****** Object:  ForeignKey [FK_produkty_kategoria]    Script Date:
08/01/2011 15:43:59 ******/
ALTER TABLE [dbo].[produkty]  WITH CHECK ADD  CONSTRAINT
[FK_produkty_kategoria] FOREIGN KEY([id_kategoria])
REFERENCES [dbo].[kategoria] ([id_kategoria])
GO
ALTER TABLE [dbo].[produkty] CHECK CONSTRAINT [FK_produkty_kategoria]
GO
/****** Object:  ForeignKey [FK_produkty_producent]    Script Date:
08/01/2011 15:43:59 ******/
ALTER TABLE [dbo].[produkty]  WITH CHECK ADD  CONSTRAINT
[FK_produkty_producent] FOREIGN KEY([id_producent])
REFERENCES [dbo].[producent] ([id_producent])
GO
ALTER TABLE [dbo].[produkty] CHECK CONSTRAINT [FK_produkty_producent]
GO
/****** Object:  ForeignKey [FK_produkty_promocja]    Script Date:
08/01/2011 15:43:59 ******/
ALTER TABLE [dbo].[produkty]  WITH CHECK ADD  CONSTRAINT
[FK_produkty_promocja] FOREIGN KEY([id_promocja])
REFERENCES [dbo].[promocja] ([id_promocji])
GO
ALTER TABLE [dbo].[produkty] CHECK CONSTRAINT [FK_produkty_promocja]
GO
/****** Object:  ForeignKey [FK_produkty_zamowienia_produkty]
Script Date: 08/01/2011 15:43:59 ******/
ALTER TABLE [dbo].[produkty_zamowienia]  WITH CHECK ADD  CONSTRAINT
[FK_produkty_zamowienia_produkty] FOREIGN KEY([id_produkty])
REFERENCES [dbo].[produkty] ([id_produkty])
GO
ALTER TABLE [dbo].[produkty_zamowienia] CHECK CONSTRAINT
[FK_produkty_zamowienia_produkty]
GO
/****** Object:  ForeignKey [FK_produkty_zamowienia_zamowienia]
Script Date: 08/01/2011 15:43:59 ******/
ALTER TABLE [dbo].[produkty_zamowienia]  WITH CHECK ADD  CONSTRAINT
[FK_produkty_zamowienia_zamowienia] FOREIGN KEY([id_zamowienia])
REFERENCES [dbo].[zamowienia] ([id_zamowienia])
GO
ALTER TABLE [dbo].[produkty_zamowienia] CHECK CONSTRAINT
[FK_produkty_zamowienia_zamowienia]
GO
```

```
/****** Object:  ForeignKey [FK_sprzedaz_kontrahenci]    Script Date:
08/01/2011 15:43:59 ******/
ALTER TABLE [dbo].[sprzedaz]  WITH CHECK ADD  CONSTRAINT
[FK_sprzedaz_kontrahenci] FOREIGN KEY([id_kontrahenci])
REFERENCES [dbo].[kontrahenci] ([id_kontrahenci])
GO
ALTER TABLE [dbo].[sprzedaz] CHECK CONSTRAINT
[FK_sprzedaz_kontrahenci]
GO
/****** Object:  ForeignKey [FK_sprzedaz_pracownik]    Script Date:
08/01/2011 15:43:59 ******/
ALTER TABLE [dbo].[sprzedaz]  WITH CHECK ADD  CONSTRAINT
[FK_sprzedaz_pracownik] FOREIGN KEY([id_pracownika])
REFERENCES [dbo].[pracownik] ([id_pracownika])
GO
ALTER TABLE [dbo].[sprzedaz] CHECK CONSTRAINT [FK_sprzedaz_pracownik]
GO
/****** Object:  ForeignKey [FK_sprzedaz_zamowienia]    Script Date:
08/01/2011 15:43:59 ******/
ALTER TABLE [dbo].[sprzedaz]  WITH CHECK ADD  CONSTRAINT
[FK_sprzedaz_zamowienia] FOREIGN KEY([id_zamowienia])
REFERENCES [dbo].[zamowienia] ([id_zamowienia])
GO
ALTER TABLE [dbo].[sprzedaz] CHECK CONSTRAINT [FK_sprzedaz_zamowienia]
GO
```

3.5 Procedury – przykłady

W rozdziale zaprezentowane zostały przykładowe procedury używane podczas pracy projektowanego systemu.

a) Procedury dotyczące tabeli adres:

```
USE [hurtownia];
GO

IF OBJECT_ID('[dbo].[usp_adresSelect]') IS NOT NULL
BEGIN
    DROP PROC [dbo].[usp_adresSelect]
END
GO
CREATE PROC [dbo].[usp_adresSelect]
    @id_adres INT
AS
      SET NOCOUNT ON
      SET XACT_ABORT ON

      BEGIN TRAN

      SELECT [id_adres], [panstwo], [miasto], [ulica], [kod_pocztowy],
[nr_domu], [nr_mieszkania], [id_wojewodztwo]
      FROM   [dbo].[adres]
      WHERE  ([id_adres] = @id_adres OR @id_adres IS NULL)

      COMMIT
GO
IF OBJECT_ID('[dbo].[usp_adresInsert]') IS NOT NULL
BEGIN
```

```
    DROP PROC [dbo].[usp_adresInsert]
END
GO
CREATE PROC [dbo].[usp_adresInsert]
    @panstwo nchar(20),
    @miasto nchar(20),
    @ulica nchar(30),
    @kod_pocztowy nchar(6),
    @nr_domu nchar(10),
    @nr_mieszkania nchar(10),
    @id_wojewodztwo int
AS
      SET NOCOUNT ON
      SET XACT_ABORT ON

      BEGIN TRAN

      INSERT INTO [dbo].[adres] ([panstwo], [miasto], [ulica],
[kod_pocztowy], [nr_domu], [nr_mieszkania], [id_wojewodztwo])
      SELECT @panstwo, @miasto, @ulica, @kod_pocztowy, @nr_domu,
@nr_mieszkania, @id_wojewodztwo

      -- Begin Return Select <- do not remove
      SELECT [id_adres], [panstwo], [miasto], [ulica], [kod_pocztowy],
[nr_domu], [nr_mieszkania], [id_wojewodztwo]
      FROM   [dbo].[adres]
      WHERE  [id_adres] = SCOPE_IDENTITY()
      -- End Return Select <- do not remove

      COMMIT
GO
IF OBJECT_ID('[dbo].[usp_adresUpdate]') IS NOT NULL
BEGIN
    DROP PROC [dbo].[usp_adresUpdate]
END
GO
CREATE PROC [dbo].[usp_adresUpdate]
    @id_adres int,
    @panstwo nchar(20),
    @miasto nchar(20),
    @ulica nchar(30),
    @kod_pocztowy nchar(6),
    @nr_domu nchar(10),
    @nr_mieszkania nchar(10),
    @id_wojewodztwo int
AS
      SET NOCOUNT ON
      SET XACT_ABORT ON

      BEGIN TRAN

      UPDATE [dbo].[adres]
      SET    [panstwo] = @panstwo, [miasto] = @miasto, [ulica] =
@ulica, [kod_pocztowy] = @kod_pocztowy, [nr_domu] = @nr_domu,
[nr_mieszkania] = @nr_mieszkania, [id_wojewodztwo] = @id_wojewodztwo
      WHERE  [id_adres] = @id_adres

      -- Begin Return Select <- do not remove
      SELECT [id_adres], [panstwo], [miasto], [ulica], [kod_pocztowy],
[nr_domu], [nr_mieszkania], [id_wojewodztwo]
      FROM   [dbo].[adres]
```

```
      WHERE  [id_adres] = @id_adres
      -- End Return Select <- do not remove

      COMMIT TRAN
GO
IF OBJECT_ID('[dbo].[usp_adresDelete]') IS NOT NULL
BEGIN
    DROP PROC [dbo].[usp_adresDelete]
END
GO
CREATE PROC [dbo].[usp_adresDelete]
    @id_adres int
AS
      SET NOCOUNT ON
      SET XACT_ABORT ON

      BEGIN TRAN

      DELETE
      FROM   [dbo].[adres]
      WHERE  [id_adres] = @id_adres

      COMMIT
     GO
```

b) Procedury dotyczące tabeli kontrahenci:

```
USE [hurtownia];
GO

IF OBJECT_ID('[dbo].[usp_kontrahenciSelect]') IS NOT NULL
BEGIN
    DROP PROC [dbo].[usp_kontrahenciSelect]
END
GO
CREATE PROC [dbo].[usp_kontrahenciSelect]
    @id_kontrahenci INT
AS
      SET NOCOUNT ON
      SET XACT_ABORT ON

      BEGIN TRAN

      SELECT [id_kontrahenci], [Nazwa], [id_adres_wysylki],
[id_adres_fv], [nip], [regon], [id_osoba_kontaktowa], [id_pracownika],
[umowa], [data_podpisania_umowy], [kredyt_kupiecki], [wartosc_kredytu]
      FROM   [dbo].[kontrahenci]
      WHERE  ([id_kontrahenci] = @id_kontrahenci OR @id_kontrahenci IS
NULL)

      COMMIT
GO
IF OBJECT_ID('[dbo].[usp_kontrahenciInsert]') IS NOT NULL
BEGIN
    DROP PROC [dbo].[usp_kontrahenciInsert]
END
GO
CREATE PROC [dbo].[usp_kontrahenciInsert]
    @id_kontrahenci int,
    @Nazwa nvarchar(30),
    @id_adres_wysylki int,
    @id_adres_fv int,
```

```
    @nip nvarchar(10),
    @regon int,
    @id_osoba_kontaktowa int,
    @id_pracownika nvarchar(7),
    @umowa bit,
    @data_podpisania_umowy datetime,
    @kredyt_kupiecki binary(1),
    @wartosc_kredytu money
AS
      SET NOCOUNT ON
      SET XACT_ABORT ON

      BEGIN TRAN

      INSERT INTO [dbo].[kontrahenci] ([id_kontrahenci], [Nazwa],
[id_adres_wysylki], [id_adres_fv], [nip], [regon],
[id_osoba_kontaktowa], [id_pracownika], [umowa],
[data_podpisania_umowy], [kredyt_kupiecki], [wartosc_kredytu])
      SELECT @id_kontrahenci, @Nazwa, @id_adres_wysylki, @id_adres_fv,
@nip, @regon, @id_osoba_kontaktowa, @id_pracownika, @umowa,
@data_podpisania_umowy, @kredyt_kupiecki, @wartosc_kredytu

      -- Begin Return Select <- do not remove
      SELECT [id_kontrahenci], [Nazwa], [id_adres_wysylki],
[id_adres_fv], [nip], [regon], [id_osoba_kontaktowa], [id_pracownika],
[umowa], [data_podpisania_umowy], [kredyt_kupiecki], [wartosc_kredytu]
      FROM   [dbo].[kontrahenci]
      WHERE  [id_kontrahenci] = @id_kontrahenci
      -- End Return Select <- do not remove

      COMMIT
GO
IF OBJECT_ID('[dbo].[usp_kontrahenciUpdate]') IS NOT NULL
BEGIN
    DROP PROC [dbo].[usp_kontrahenciUpdate]
END
GO
CREATE PROC [dbo].[usp_kontrahenciUpdate]
    @id_kontrahenci int,
    @Nazwa nvarchar(30),
    @id_adres_wysylki int,
    @id_adres_fv int,
    @nip nvarchar(10),
    @regon int,
    @id_osoba_kontaktowa int,
    @id_pracownika nvarchar(7),
    @umowa bit,
    @data_podpisania_umowy datetime,
    @kredyt_kupiecki binary(1),
    @wartosc_kredytu money
AS
      SET NOCOUNT ON
      SET XACT_ABORT ON

      BEGIN TRAN

      UPDATE [dbo].[kontrahenci]
      SET    [id_kontrahenci] = @id_kontrahenci, [Nazwa] = @Nazwa,
[id_adres_wysylki] = @id_adres_wysylki, [id_adres_fv] = @id_adres_fv,
[nip] = @nip, [regon] = @regon, [id_osoba_kontaktowa] =
@id_osoba_kontaktowa, [id_pracownika] = @id_pracownika, [umowa] =
```

```
@umowa, [data_podpisania_umowy] = @data_podpisania_umowy,
[kredyt_kupiecki] = @kredyt_kupiecki, [wartosc_kredytu] =
@wartosc_kredytu
      WHERE  [id_kontrahenci] = @id_kontrahenci

      -- Begin Return Select <- do not remove
      SELECT [id_kontrahenci], [Nazwa], [id_adres_wysylki],
[id_adres_fv], [nip], [regon], [id_osoba_kontaktowa], [id_pracownika],
[umowa], [data_podpisania_umowy], [kredyt_kupiecki], [wartosc_kredytu]
      FROM   [dbo].[kontrahenci]
      WHERE  [id_kontrahenci] = @id_kontrahenci
      -- End Return Select <- do not remove

      COMMIT TRAN
GO
IF OBJECT_ID('[dbo].[usp_kontrahenciDelete]') IS NOT NULL
BEGIN
    DROP PROC [dbo].[usp_kontrahenciDelete]
END
GO
CREATE PROC [dbo].[usp_kontrahenciDelete]
    @id_kontrahenci int
AS
      SET NOCOUNT ON
      SET XACT_ABORT ON

      BEGIN TRAN

      DELETE
      FROM   [dbo].[kontrahenci]
      WHERE  [id_kontrahenci] = @id_kontrahenci

      COMMIT
GO
```

c) Procedury dotyczące tabeli sprzedaż:

```
USE [hurtownia];
GO

IF OBJECT_ID('[dbo].[usp_sprzedazSelect]') IS NOT NULL
BEGIN
    DROP PROC [dbo].[usp_sprzedazSelect]
END
GO
CREATE PROC [dbo].[usp_sprzedazSelect]
    @id_sprzedaz INT
AS
      SET NOCOUNT ON
      SET XACT_ABORT ON

      BEGIN TRAN

      SELECT [id_sprzedaz], [id_zamowienia], [id_pracownika],
[id_kontrahenci]
      FROM   [dbo].[sprzedaz]
      WHERE  ([id_sprzedaz] = @id_sprzedaz OR @id_sprzedaz IS NULL)

      COMMIT
GO
IF OBJECT_ID('[dbo].[usp_sprzedazInsert]') IS NOT NULL
BEGIN
```

```
    DROP PROC [dbo].[usp_sprzedazInsert]
END
GO
CREATE PROC [dbo].[usp_sprzedazInsert]
    @id_zamowienia int,
    @id_pracownika nvarchar(7),
    @id_kontrahenci int
AS
      SET NOCOUNT ON
      SET XACT_ABORT ON

      BEGIN TRAN

      INSERT INTO [dbo].[sprzedaz] ([id_zamowienia], [id_pracownika],
[id_kontrahenci])
      SELECT @id_zamowienia, @id_pracownika, @id_kontrahenci

      -- Begin Return Select <- do not remove
      SELECT [id_sprzedaz], [id_zamowienia], [id_pracownika],
[id_kontrahenci]
      FROM   [dbo].[sprzedaz]
      WHERE  [id_sprzedaz] = SCOPE_IDENTITY()
      -- End Return Select <- do not remove

      COMMIT
GO
IF OBJECT_ID('[dbo].[usp_sprzedazUpdate]') IS NOT NULL
BEGIN
    DROP PROC [dbo].[usp_sprzedazUpdate]
END
GO
CREATE PROC [dbo].[usp_sprzedazUpdate]
    @id_sprzedaz int,
    @id_zamowienia int,
    @id_pracownika nvarchar(7),
    @id_kontrahenci int
AS
      SET NOCOUNT ON
      SET XACT_ABORT ON

      BEGIN TRAN

      UPDATE [dbo].[sprzedaz]
      SET    [id_zamowienia] = @id_zamowienia, [id_pracownika] =
@id_pracownika, [id_kontrahenci] = @id_kontrahenci
      WHERE  [id_sprzedaz] = @id_sprzedaz

      -- Begin Return Select <- do not remove
      SELECT [id_sprzedaz], [id_zamowienia], [id_pracownika],
[id_kontrahenci]
      FROM   [dbo].[sprzedaz]
      WHERE  [id_sprzedaz] = @id_sprzedaz
      -- End Return Select <- do not remove

      COMMIT TRAN
GO
IF OBJECT_ID('[dbo].[usp_sprzedazDelete]') IS NOT NULL
BEGIN
    DROP PROC [dbo].[usp_sprzedazDelete]
END
GO
```

```
CREATE PROC [dbo].[usp_sprzedazDelete]
    @id_sprzedaz int
AS
      SET NOCOUNT ON
      SET XACT_ABORT ON

      BEGIN TRAN

      DELETE
      FROM   [dbo].[sprzedaz]
      WHERE  [id_sprzedaz] = @id_sprzedaz

      COMMIT
GO
```

d) Procedury dotyczące tabeli produkty:

```
USE [hurtownia];
GO

IF OBJECT_ID('[dbo].[usp_produktySelect]') IS NOT NULL
BEGIN
    DROP PROC [dbo].[usp_produktySelect]
END
GO
CREATE PROC [dbo].[usp_produktySelect]
    @id_produkty INT
AS
      SET NOCOUNT ON
      SET XACT_ABORT ON

      BEGIN TRAN

      SELECT [id_produkty], [id_producent], [model], [opis],
[cena_katalogowa], [stan_magazynowy], [id_kategoria], [PN], [aktywne],
[id_promocja]
      FROM   [dbo].[produkty]
      WHERE  ([id_produkty] = @id_produkty OR @id_produkty IS NULL)

      COMMIT
GO
IF OBJECT_ID('[dbo].[usp_produktyInsert]') IS NOT NULL
BEGIN
    DROP PROC [dbo].[usp_produktyInsert]
END
GO
CREATE PROC [dbo].[usp_produktyInsert]
    @id_producent int,
    @model nchar(30),
    @opis varchar(MAX),
    @cena_katalogowa decimal(19, 4),
    @stan_magazynowy int,
    @id_kategoria int,
    @PN nvarchar(20),
    @aktywne binary(1),
    @id_promocja int
AS
      SET NOCOUNT ON
      SET XACT_ABORT ON

      BEGIN TRAN
```

```
      INSERT INTO [dbo].[produkty] ([id_producent], [model], [opis],
[cena_katalogowa], [stan_magazynowy], [id_kategoria], [PN], [aktywne],
[id_promocja])
      SELECT @id_producent, @model, @opis, @cena_katalogowa,
@stan_magazynowy, @id_kategoria, @PN, @aktywne, @id_promocja

      -- Begin Return Select <- do not remove
      SELECT [id_produkty], [id_producent], [model], [opis],
[cena_katalogowa], [stan_magazynowy], [id_kategoria], [PN], [aktywne],
[id_promocja]
      FROM   [dbo].[produkty]
      WHERE  [id_produkty] = SCOPE_IDENTITY()
      -- End Return Select <- do not remove

      COMMIT
GO
IF OBJECT_ID('[dbo].[usp_produktyUpdate]') IS NOT NULL
BEGIN
    DROP PROC [dbo].[usp_produktyUpdate]
END
GO
CREATE PROC [dbo].[usp_produktyUpdate]
    @id_produkty int,
    @id_producent int,
    @model nchar(30),
    @opis varchar(MAX),
    @cena_katalogowa decimal(19, 4),
    @stan_magazynowy int,
    @id_kategoria int,
    @PN nvarchar(20),
    @aktywne binary(1),
    @id_promocja int
AS
      SET NOCOUNT ON
      SET XACT_ABORT ON

      BEGIN TRAN

      UPDATE [dbo].[produkty]
      SET    [id_producent] = @id_producent, [model] = @model, [opis]
= @opis, [cena_katalogowa] = @cena_katalogowa, [stan_magazynowy] =
@stan_magazynowy, [id_kategoria] = @id_kategoria, [PN] = @PN,
[aktywne] = @aktywne, [id_promocja] = @id_promocja
      WHERE  [id_produkty] = @id_produkty

      -- Begin Return Select <- do not remove
      SELECT [id_produkty], [id_producent], [model], [opis],
[cena_katalogowa], [stan_magazynowy], [id_kategoria], [PN], [aktywne],
[id_promocja]
      FROM   [dbo].[produkty]
      WHERE  [id_produkty] = @id_produkty
      -- End Return Select <- do not remove

      COMMIT TRAN
GO
IF OBJECT_ID('[dbo].[usp_produktyDelete]') IS NOT NULL
BEGIN
    DROP PROC [dbo].[usp_produktyDelete]
END
GO
CREATE PROC [dbo].[usp_produktyDelete]
```

```
    @id_produkty int
AS
      SET NOCOUNT ON
      SET XACT_ABORT ON

      BEGIN TRAN

      DELETE
      FROM   [dbo].[produkty]
      WHERE  [id_produkty] = @id_produkty

      COMMIT
GO
```

e) Procedury dotyczące tabeli pracownik:

```
USE [hurtownia];
GO

IF OBJECT_ID('[dbo].[usp_pracownikSelect]') IS NOT NULL
BEGIN
    DROP PROC [dbo].[usp_pracownikSelect]
END
GO
CREATE PROC [dbo].[usp_pracownikSelect]
    @id_pracownika NVARCHAR(7)
AS
      SET NOCOUNT ON
      SET XACT_ABORT ON

      BEGIN TRAN

      SELECT [id_pracownika], [imie], [nazwisko], [pesel], [id_adres],
[wyplata], [pracuje]
      FROM   [dbo].[pracownik]
      WHERE  ([id_pracownika] = @id_pracownika OR @id_pracownika IS
NULL)

      COMMIT
GO
IF OBJECT_ID('[dbo].[usp_pracownikInsert]') IS NOT NULL
BEGIN
    DROP PROC [dbo].[usp_pracownikInsert]
END
GO
CREATE PROC [dbo].[usp_pracownikInsert]
    @id_pracownika nvarchar(7),
    @imie nchar(30),
    @nazwisko nchar(30),
    @pesel char(11),
    @id_adres int,
    @wyplata money,
    @pracuje binary(1)
AS
      SET NOCOUNT ON
      SET XACT_ABORT ON

      BEGIN TRAN

      INSERT INTO [dbo].[pracownik] ([id_pracownika], [imie],
[nazwisko], [pesel], [id_adres], [wyplata], [pracuje])
```

```
      SELECT @id_pracownika, @imie, @nazwisko, @pesel, @id_adres,
@wyplata, @pracuje

      -- Begin Return Select <- do not remove
      SELECT [id_pracownika], [imie], [nazwisko], [pesel], [id_adres],
[wyplata], [pracuje]
      FROM   [dbo].[pracownik]
      WHERE  [id_pracownika] = @id_pracownika
      -- End Return Select <- do not remove

      COMMIT
GO
IF OBJECT_ID('[dbo].[usp_pracownikUpdate]') IS NOT NULL
BEGIN
    DROP PROC [dbo].[usp_pracownikUpdate]
END
GO
CREATE PROC [dbo].[usp_pracownikUpdate]
    @id_pracownika nvarchar(7),
    @imie nchar(30),
    @nazwisko nchar(30),
    @pesel char(11),
    @id_adres int,
    @wyplata money,
    @pracuje binary(1)
AS
      SET NOCOUNT ON
      SET XACT_ABORT ON

      BEGIN TRAN

      UPDATE [dbo].[pracownik]
      SET    [id_pracownika] = @id_pracownika, [imie] = @imie,
[nazwisko] = @nazwisko, [pesel] = @pesel, [id_adres] = @id_adres,
[wyplata] = @wyplata, [pracuje] = @pracuje
      WHERE  [id_pracownika] = @id_pracownika

      -- Begin Return Select <- do not remove
      SELECT [id_pracownika], [imie], [nazwisko], [pesel], [id_adres],
[wyplata], [pracuje]
      FROM   [dbo].[pracownik]
      WHERE  [id_pracownika] = @id_pracownika
      -- End Return Select <- do not remove

      COMMIT TRAN
GO
IF OBJECT_ID('[dbo].[usp_pracownikDelete]') IS NOT NULL
BEGIN
    DROP PROC [dbo].[usp_pracownikDelete]
END
GO
CREATE PROC [dbo].[usp_pracownikDelete]
    @id_pracownika nvarchar(7)
AS
      SET NOCOUNT ON
      SET XACT_ABORT ON

      BEGIN TRAN

      DELETE
      FROM   [dbo].[pracownik]
```

```
        WHERE   [id_pracownika] = @id_pracownika

        COMMIT
GO
```

4 Zasada działania

Niniejszy rozdział zawiera opis i implementację wybranych elementów projektowanego systemu.

4.1 Logowanie

Po uruchomieniu aplikacji pojawia się okno główne, które pozostaje nieaktywne oraz okno logowania, które zostanie zrealizowane w trakcie dalszej pracy nad projektem, poniżej zostało jednak opisane jego działanie.

W celu zalogowania użytkownik podaje login i hasło i naciska przycisk zaloguj. Funkcja obsługująca ten przycisk za pomocą funkcji szyfrującej (Hasher) szyfruje hasło, które zostaje porównane z odpowiednim hasłem zapisanym w bazie danych w postaci skrótu. W przypadku uzyskania zgodności zostają wczytane role użytkownika. Menu główne i podmenu są takie same dla wszystkich użytkowników w zależności od roli zalogowany użytkownik będzie miał aktywne poszczególne przyciski.

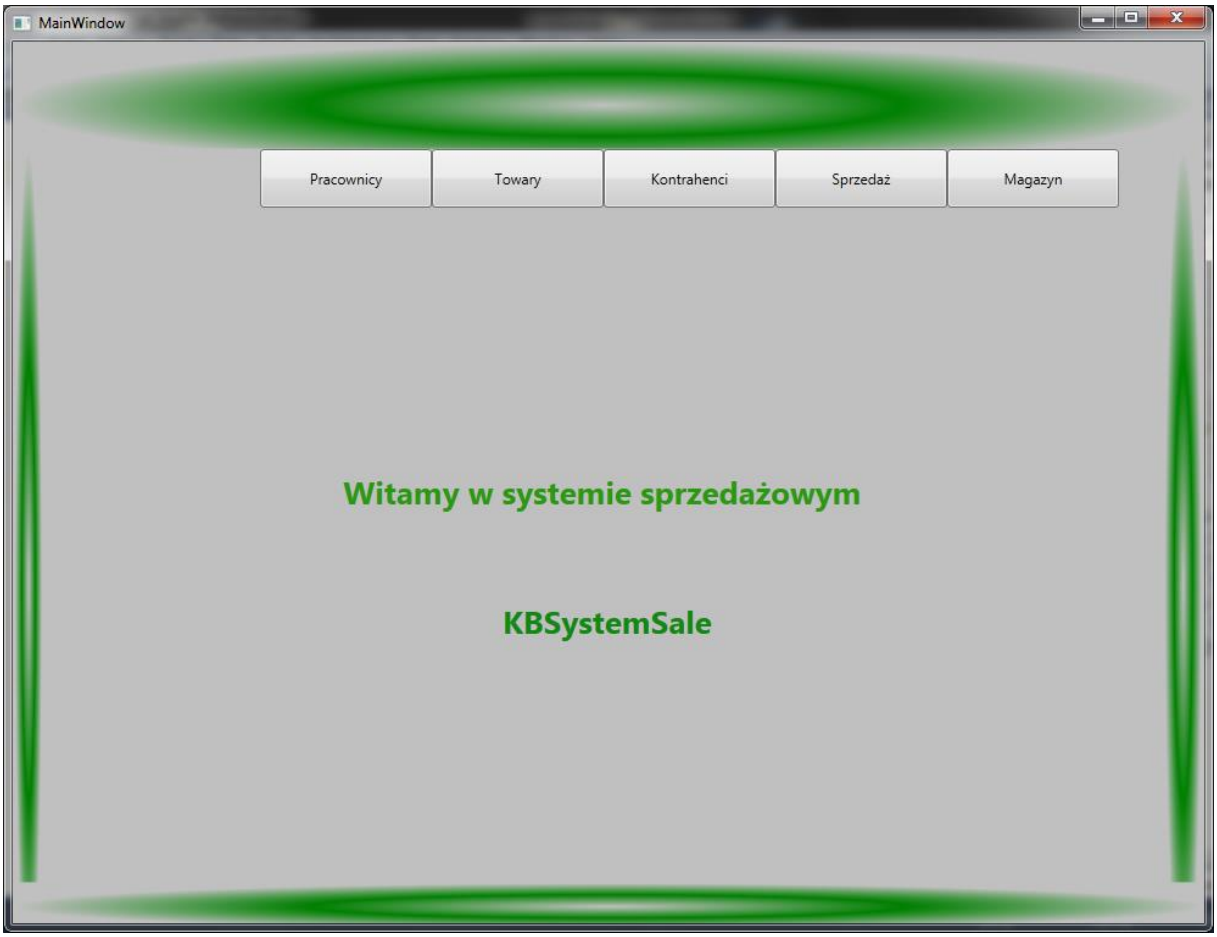
W celu łatwiejszego identyfikowania pracowników, w jakim dziale pracują, w jaki mieście login, a tym samym id_pracownika składa się z 7 znaków, które podzielone są na 3 części:

- pierwszy znak – litera oznaczająca miasto, w którym pracownik jest zatrudniony (np. w – Warszawa, k – Kraków) – zakładam, że hurtownia ma swoje oddziały w większych miastach.

- kolejne 3 znaki – to litery oznaczające dział, w którym pracuje dany pracownik (np. Zarząd – zzz, Magazynier – lgm, Księgowość – fik, Handlowiec – spr, Product Manager – zpr, Administrator – adm)

- kolejne 3 znaki – to cyfry porządkowe identyfikujące jednoznacznie pracownika.

4.2 Moduły i aktorzy korzystający z systemu



Rysunek 13 Widok menu głównego systemu
Źródło: opracowanie własne

Jak widać na aplikacja składa się z następujących modułów:

- pracownicy,
- towary,
- kontrahenci,
- sprzedaż
- magazyn,

do których dostęp mają pracownicy poszczególnych działów.

Moduł ten zawiera informację o osobach zatrudnionych w hurtowni, ma za zadanie ułatwić komunikację między pracownikami, handlowiec w przypadku negocjacji cen lub zapytania o dostępność może wyszukać Product managera odpowiedzialnego za ten rodzaj produktów.

Kolejny moduł, zawiera informacje o towarach sprzedawanych przez hurtownię, w zależności od uprawnień (które zostaną opisane dalej) pracownik może tu znaleźć

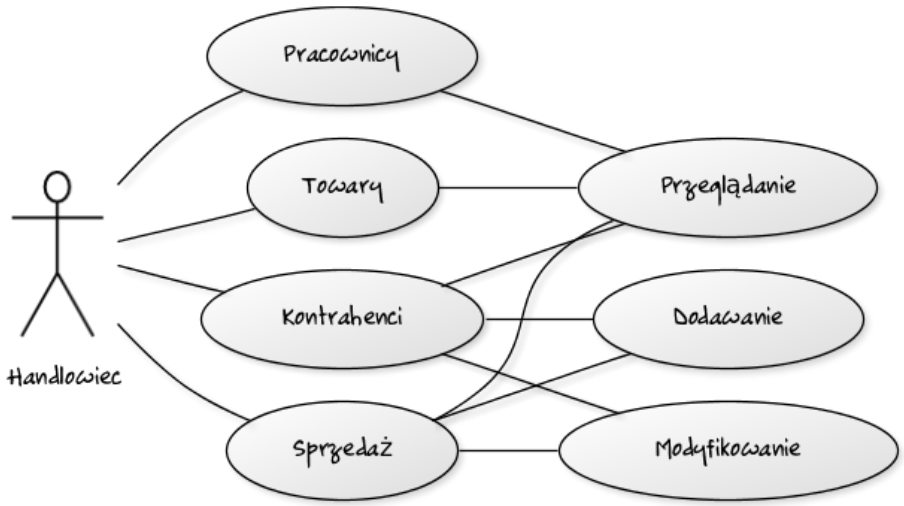
interesujący go produkt, sprawdzić cenę, ilość datę najbliższej dostawy, a także dodawać, zmieniać i zamawiać produkty.

Kontrahenci, moduł ten zawiera informacje o kontrahentach współpracujących z hurtownią. Kontrahentami mogą być zarówno dostawcy jak i odbiorcy. Moduł daje możliwość przeglądania kontrahentów (ich adresów, danych kontaktowych itp.), zmianę ich danych oraz dodawanie nowych.

Kolejnym modułem jest sprzedaż, jest to jeden z najważniejszych modułów aplikacji, zawiera informacje o zleceniach i historii sprzedaży. W tym module można tworzyć oraz przeglądać zlecenia sprzedaży jak również generować faktury sprzedaży. W momencie zamawiania towaru przez klienta generowane jest zamówienie, dopiero po potwierdzeniu, że wszystkie produkty są dostępne i klient zgadza się na ceny w zamówieniu generowana jest faktura i wysyłany jest towar.

Ostatnim modułem jest magazyn. Znajdują się tu informacje o zakupach hurtowni, oraz o zleceniach zakupów.

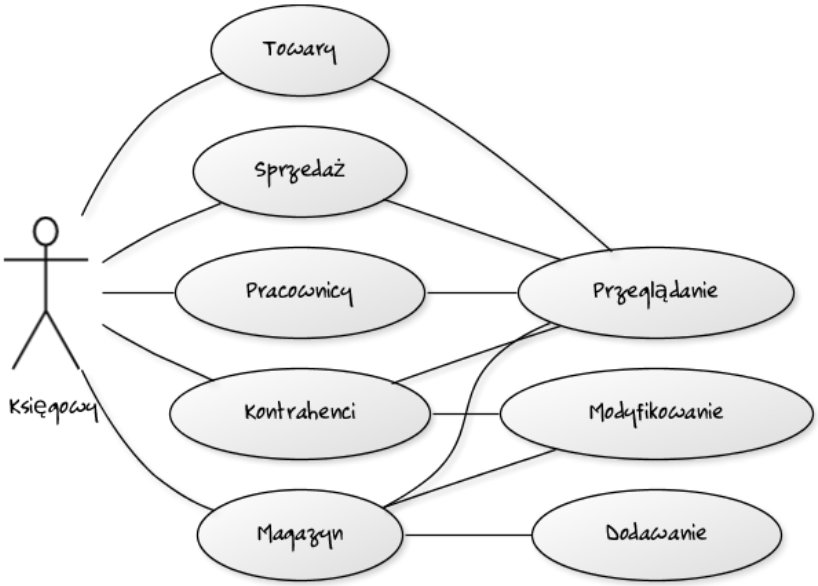
Dostęp pracowników poszczególnych działów do modułów odbywa się za pomocą loginu i hasła oraz z poziomu samej bazy danych. Uprawnienia poszczególnych grup pracowników z poziomu interfejsu opisują poniższe diagramy.



Rysunek 14 Diagram dostępu w aplikacji - Handlowiec
Źródło: opracowanie własne

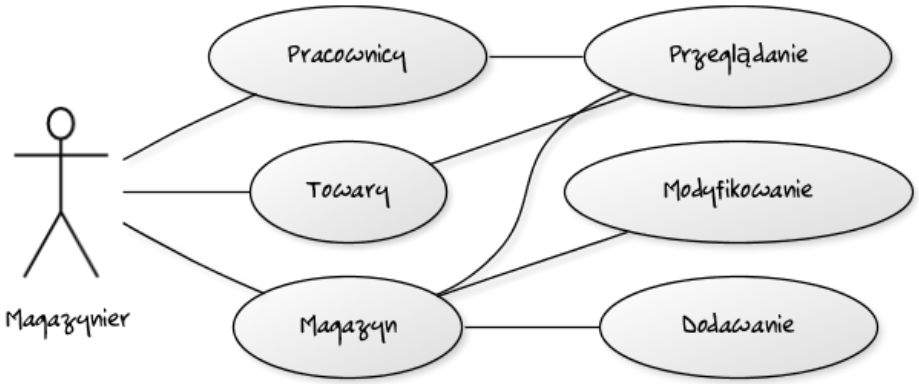
Handlowiec może wyszukiwać informacje na temat innych pracujących pracowników nie ma jednak dostępu do całości ich danych, czyli adresu zamieszkania, czy pensji. Podobnie w przypadku produktów, osoba pracująca na stanowisku handlowca może tylko i wyłącznie przeglądać produkty, nie może zmieniać ich cen, wprowadzać

nowych czy sprawdzać szczegółów. Handlowiec ma przede wszystkim tworzyć zlecenia sprzedaży, faktur sprzedaży oraz zajmować się kontrahentami w sensie klientów. Dodawać nowych, modyfikować w razie zmian i przeglądać w celu wyszukiwania interesującego go klienta.



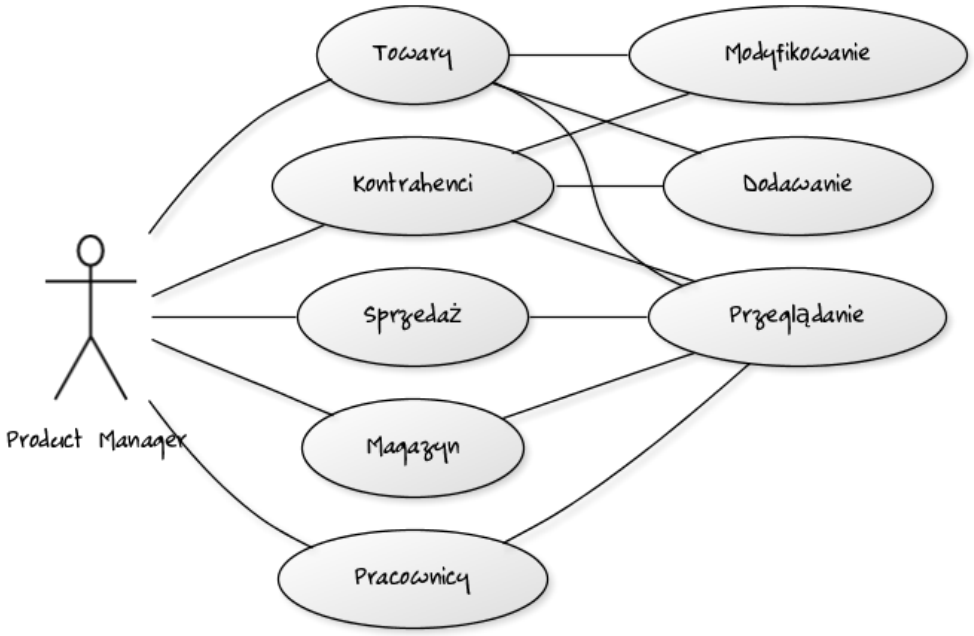
Rysunek 15 Diagram dostępu w aplikacji - Księgowy
Źródło: opracowanie własne

Księgowy ma możliwość przeglądania w ograniczonym zakresie pracowników podobnie do handlowca, ma również możliwość przeglądania faktur i zleceń sprzedaży, oraz towarów. Może modyfikować i przeglądać zlecenia zakupów, kontrahentów zmieniając informację czy dany klient ma kredyt oraz w jakiej wysokości, może również wprowadzać FV zakupu za towar, który zostanie przyjęty na magazyn.



Rysunek 16 Diagram dostępu w aplikacji - Magazynier
Źródło: opracowanie własne

Zadaniem magazyniera jest przede wszystkim sprawdzanie kompletności dostaw oraz informacji czy towar, który przyjechał został wcześniej zamówiony przez dowolnego product managera. Dodatkowo może przeglądać pracowników oraz towary.

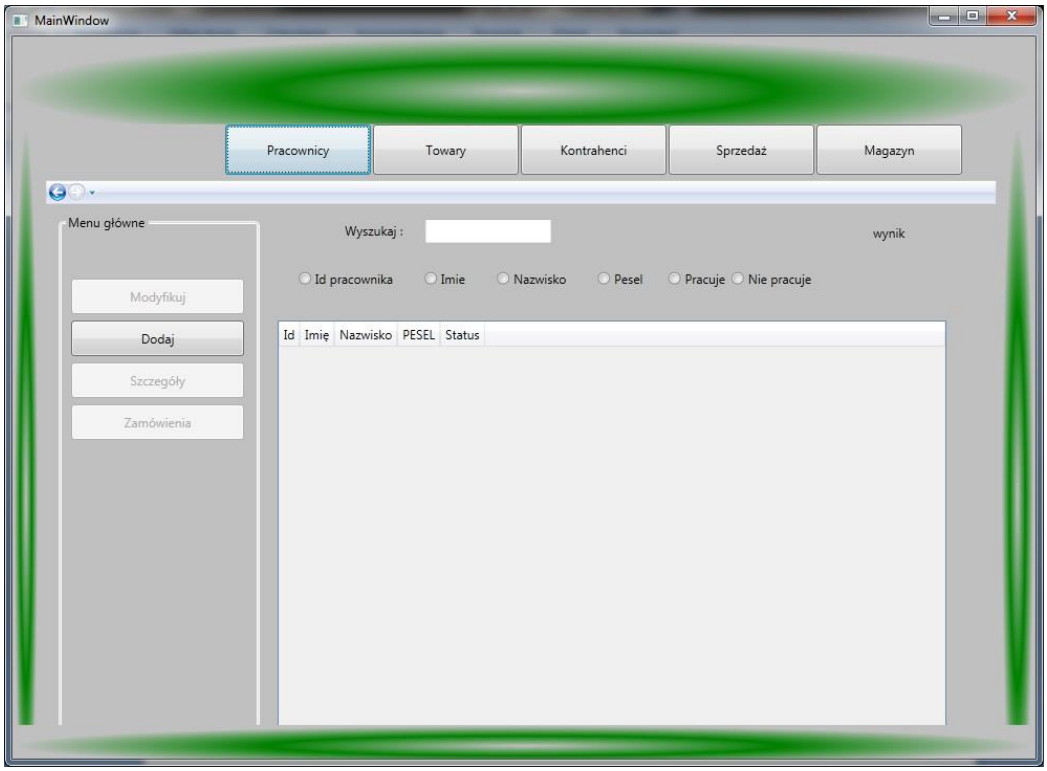


Rysunek 17 Diagram dostępu w aplikacji – Product Manager
Źródło: opracowanie własne

Product manager odpowiada przede wszystkim za ofertę hurtowni. Dodaje i modyfikuje ofertę towarową tak, aby była ona aktualna i atrakcyjna również pod względem cenowym. Pilnuje, aby towar był dostępny na magazynie tworząc zamówienia oraz aby była stała rotacja towaru. Zajmuje się również dodawaniem i modyfikowaniem kontrahentów w sensie dostawców.

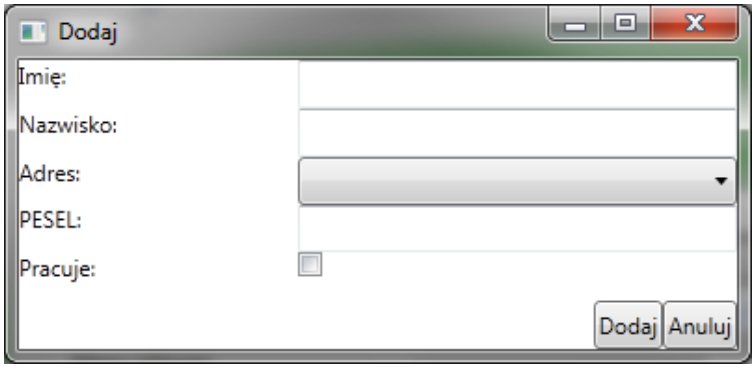
4.2.1 Moduł Pracownicy

Do modułu tego ma dostęp każdy pracownik w ograniczonym zakresie, a mianowicie tylko administrator i zarząd widzą dane osobowe pracowników, pozostałym pracownikom wyświetlane są tylko informacje dotyczące sposobu kontaktu z daną zatrudnioną osobą oraz informacje na temat tego, jakie stanowisko oraz jakimi produktami się zajmuje. Z perspektywy Administratora oraz Zarządu okno Pracownicy wygląda następująco:



Rysunek 18 Widok modułu pracownicy
Źródło: opracowanie własne

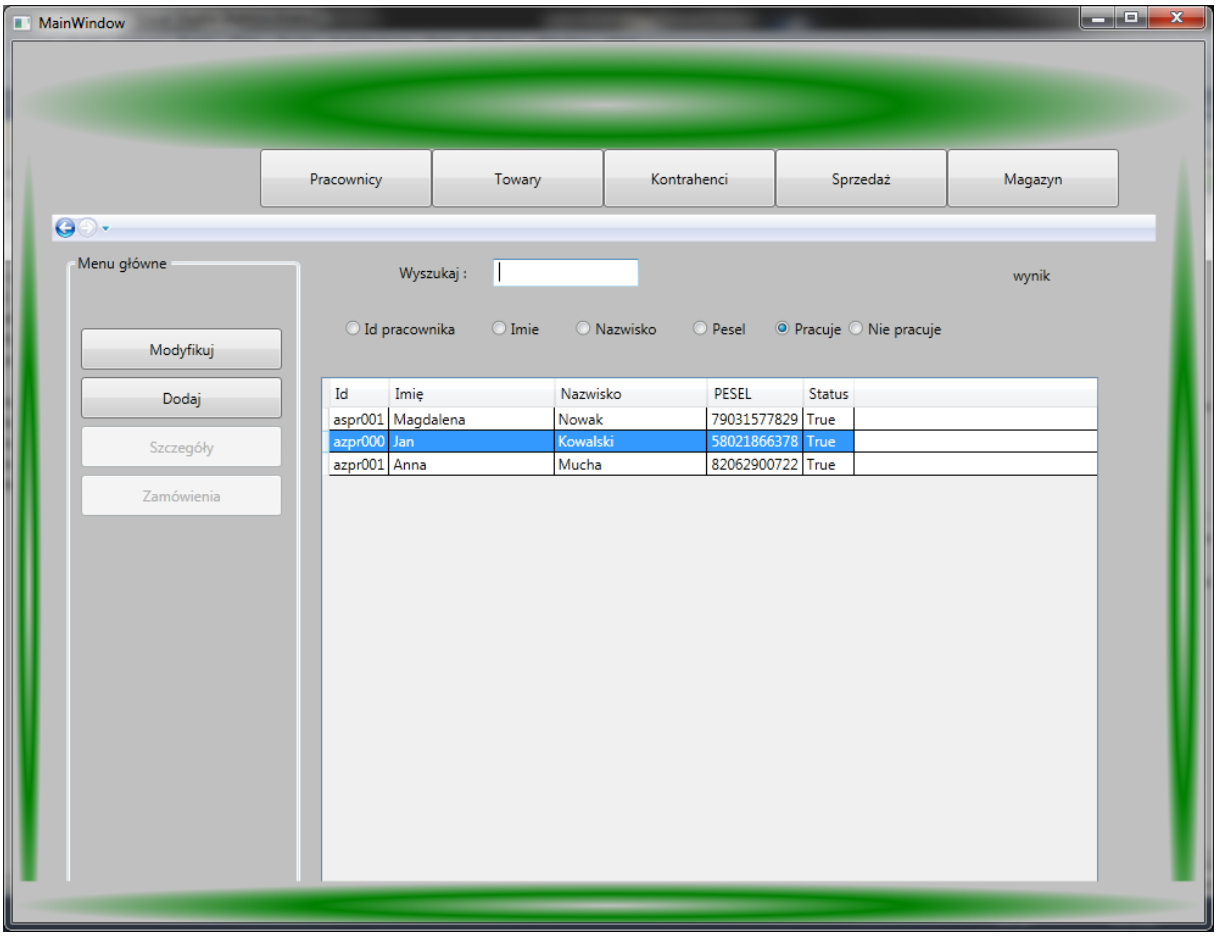
W pierwszym oknie Administrator może dodać nową osobę wówczas pojawia się dodatkowe okno:



Rysunek 19 Widok okna dodawania nowego pracownika
Źródło: opracowanie własne

Tutaj administrator może wprowadzić dane nowego pracownika w celu dodania go do bazy.

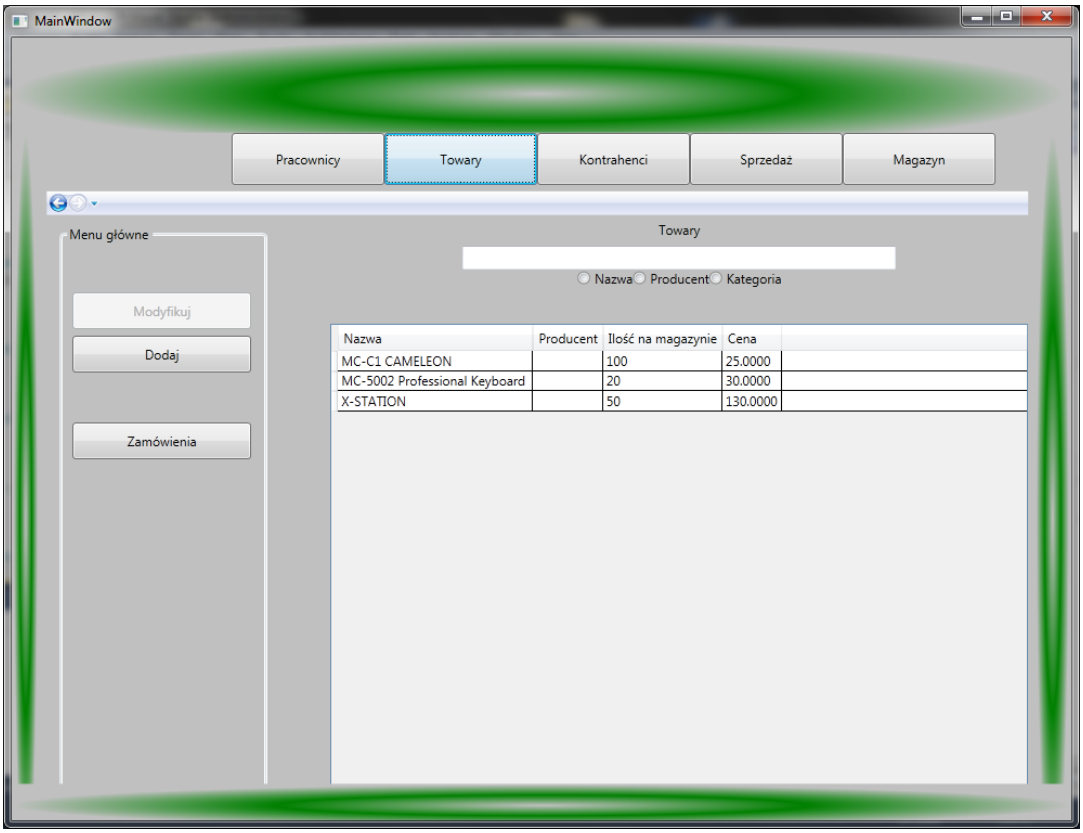
Po wybraniu interesującego nas pracownika i kliknięciu na nim w celu zaznaczenia go odblokowuje się kolejny przycisk „Modyfikuj”, gdzie można przeglądać szczegóły dotyczące wybranego pracownika.



Rysunek 20 Widok modułu pracownicy
Źródło: opracowanie własne

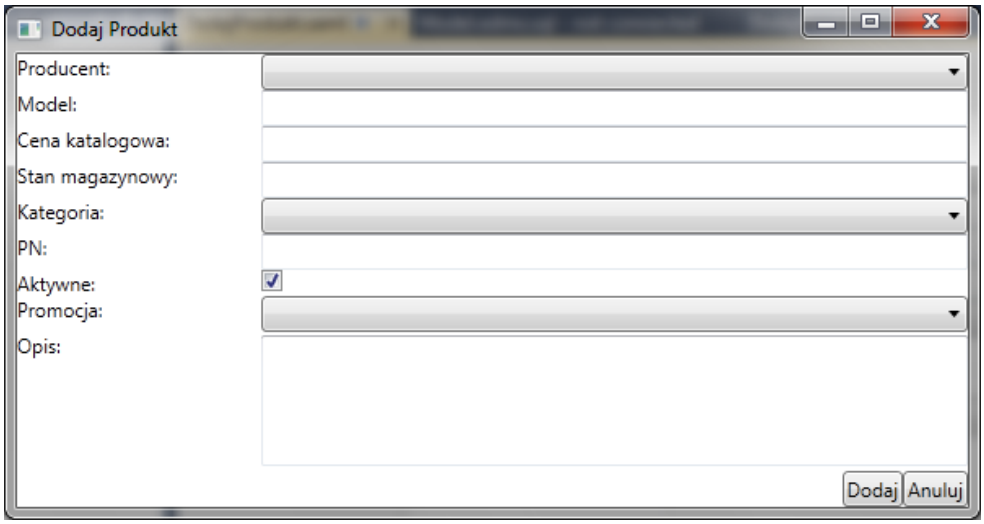
4.2.2 Moduł Towary

Moduł towary jest dostępny dla każdego pracownika w celu przeglądania towarów i sprawdzania cen. Nowe produkty prócz administratora i zarządu, może dodawać tylko Product manager. To on zajmuje się wprowadzaniem nowych produktów, ustalaniem cen, promocji zamawianiem towarów (czyli tworzeniem zleceń zakupów). Z jego perspektywy okno Towary wygląda następująco:



Rysunek 21 Widok modułu Towary
Źródło: opracowanie własne

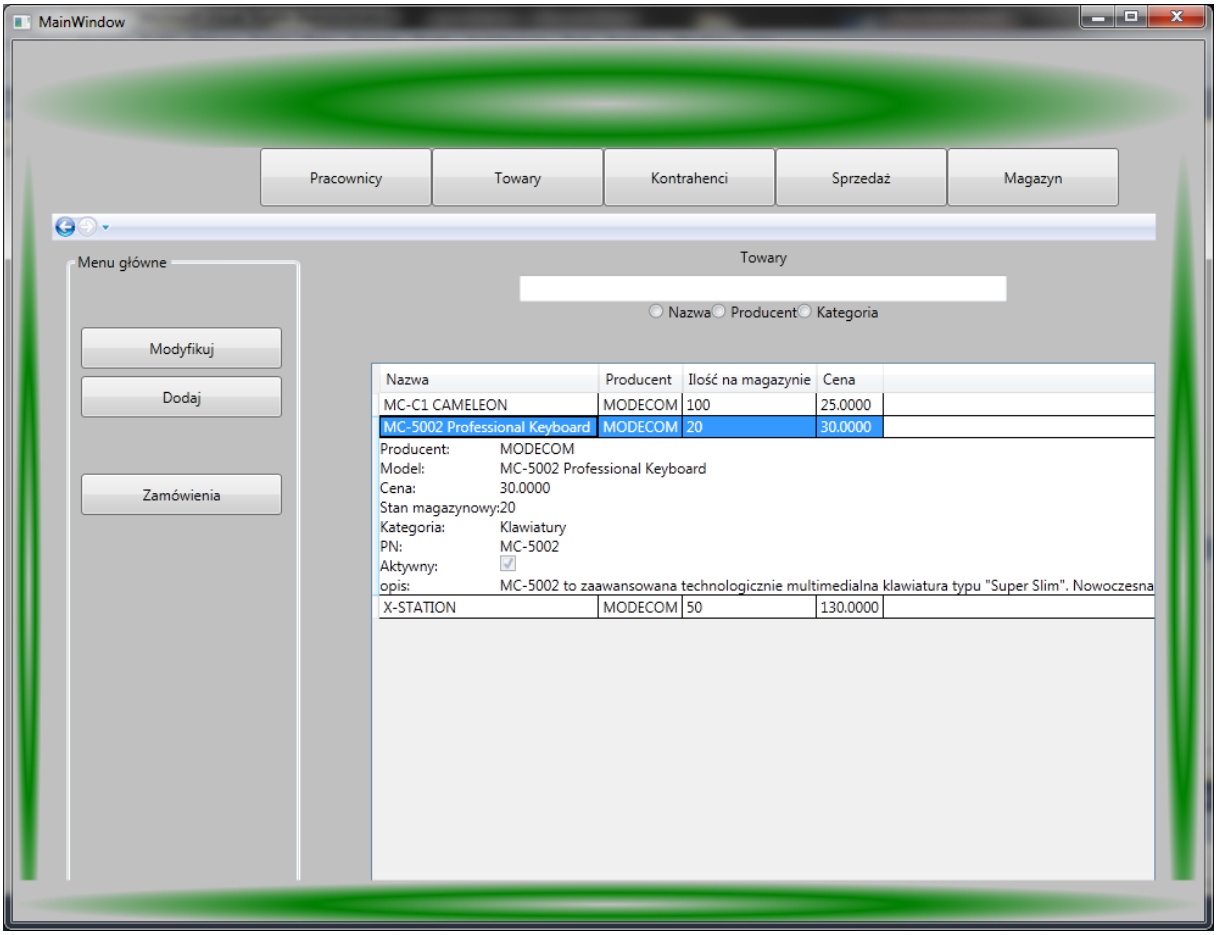
W pierwszym oknie Product Manager może dodać nowy produkt wówczas pojawia się dodatkowe okno:



Rysunek 22 Widok okna dodawania nowego produktu
Źródło: opracowanie własne

Po wypełnieniu okna i naciśnięciu przycisku „dodaj” produkt zostanie dodany do bazy.

Kolejną możliwość, jaką ma Product Manager to możliwość przeglądania szczegółów i modyfikację informacji dotyczących wybranego produktu, w tym celu należy zaznaczyć interesujący produkt i automatycznie pojawią się szczegóły:



Rysunek 23 Widok okna szczegółów produktów
Źródło: opracowanie własne

Modyfikowanie odbywa się analogicznie do modyfikacji pracowników, czyli po wybraniu interesującego nas produktu można kliknąć przycisk Modyfikuj i wówczas otwiera się okno z informacjami dotyczącymi szczegółów produktów, które możemy w dowolny sposób zmieniać.

5 Podsumowanie i wnioski

Głównym celem niniejszej pracy było stworzenie projektu i implementacji aplikacji bazodanowej, wspomagającej pracę hurtowni sprzętu IT. W oparciu o przedstawione technologie (m.in. MS SQL Server, teoria relacyjnych baz danych) został wykonany projekt systemu, zawierający zebrane wymagania funkcjonalne i niefunkcjonalne, koncepcję działania i schemat logiczny. W rozdziale poświęconym projektowi bazy danych przedstawiony został model konceptualny i fizyczny oraz omówiony został aspekt bezpieczeństwa i dostępu do danych. Następnie zrealizowana została baza danych i odpowiednie procedury składowane. W części poświęconej zasadom działania aplikacji zaprezentowane zostały wygląd i działanie zrealizowanych modułów. Objęły one zagadnienia związane z osobami zatrudnionymi w przedsiębiorstwie (moduł Pracownicy) oraz z asortymentem oferowanym przez hurtownię (moduł Towary). W modułach tych zrealizowane zostały funkcjonalności związane z wyświetlaniem, dodawaniem oraz modyfikowaniem wybranych pracowników czy towarów. Należy także zaznaczyć, że możliwa jest stosunkowo szybka i prosta rozbudowa aplikacji o kolejne moduły: Kontrahenci, Sprzedaż oraz Magazyn.

Cały system powstał w technologii Windows Presentation Foundation, która jest technologią o ogromnym potencjale i wielorakich możliwościach rozwoju. Wobec powyższego, główny cel niniejszej pracy można uznać, jako osiągnięty.

Bibliografia

1. Lee W.M., *C# 2008. Warsztat programisty*, Helion, Gliwice, 2010
2. Troelsen A., *Język C# 2008 i platforma NET 3.5*, PWN, Warszawa, 2009
3. Micah M., Robert C.M., *Agile. Programowanie zwinne: zasady, wzorce i praktyki zwinnego wytwarzania oprogramowania w C#*, Helion, Gliwice, 2008
4. Sarkar D., Katibah E., LOW G., Ben-Gan I., Kunen I., Wolter R., *Microsoft SQL Server 2008 od środka Programowanie w języku T-SQL*, Microsoft Press, 2010
5. Judith S.B., Darnovsky M., Emerson S.L., *Podręcznik języka SQL*, WNT, 2001

Internet: http://msdn.microsoft.com/pl-pl/, 05.02.2011

Internet: http://www.centrumxp.pl/, 05.02.2011

Spis tabel

Spis rysunków