Dokumentacja Bazy Danych

Autorzy

Patryk Midera, Paweł Prochot

Spis treści

Cel i założenia

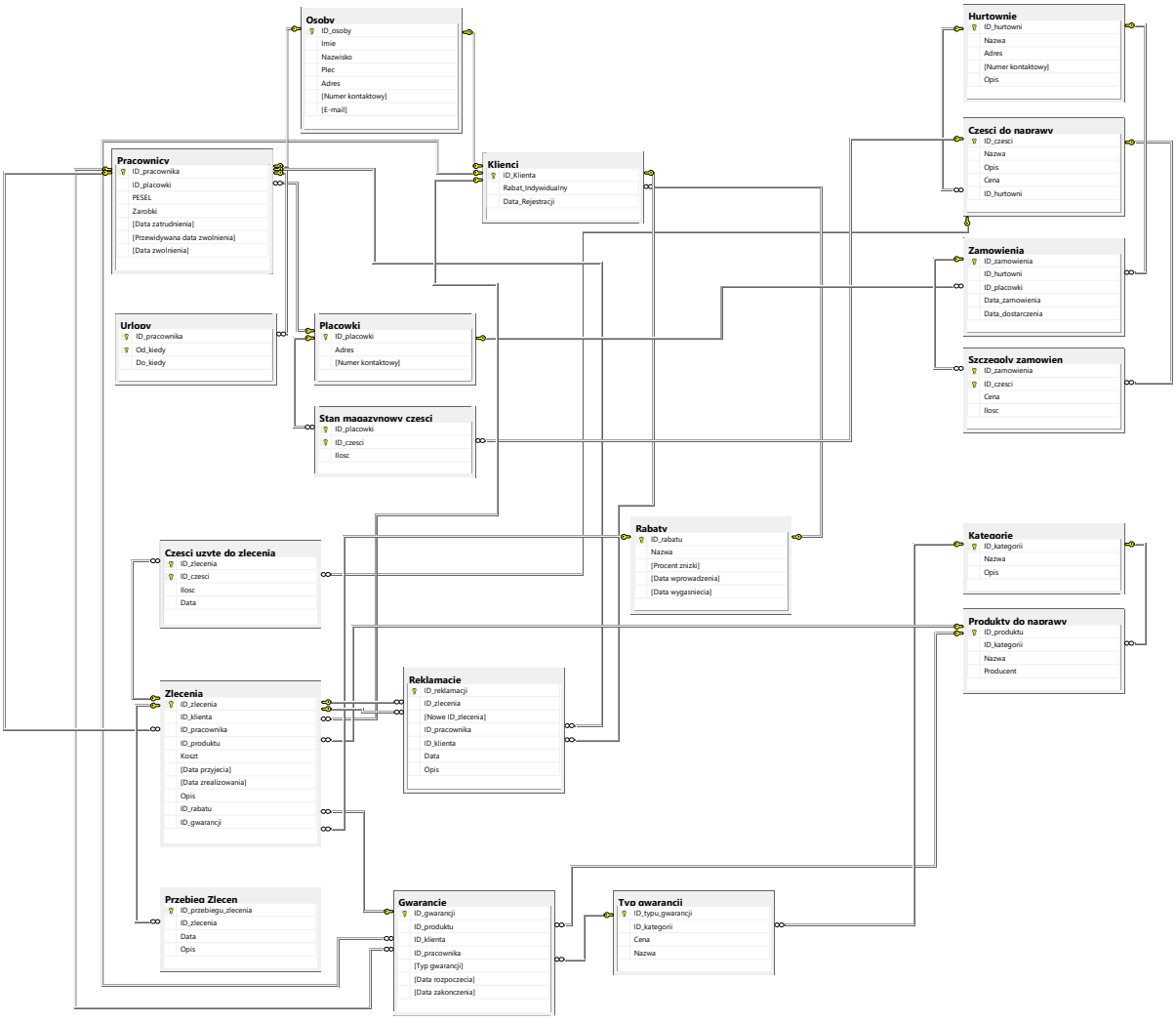
- Celem tego projektu jest stworzenie Bazy Danych gromadzącej i przetwarzającej informacji na temat funkcjonowania Serwisu naprawczego sprzętów RTV.
- **Jakiego typu informacje przechowuje**?
    - Podstawowe informacje o pracownikach i klientach korzystających z serwisu naprawczego, oraz o placówkach posiadanych przez serwis.
    - Informacje o częściach których używa się do naprawy zleconych towarów, ich zaopatrzeniowcach (hurtownie) oraz historię ich zamówień.
    - Historię zleceń i ich przebieg.
    - Informacje o naprawianych towarach.
    - Informacje o gwarancjach zakupionych na dany towar.
    - Różne rabaty dla klientów, urlopy pracowników itd.
- **Możliwości**:
    - Pozwala monitorować przebieg zleceń.
    - Zezwala na składanie reklamacji na źle wykonaną naprawę urządzenia.
    - Pozwala tworzyć różnego typu gwarancje.
    - Gromadzi informacje o urlopach pracowników.
    - Samoczynnie dokonuje „zamagazynowania” części do napraw, po dostarczeniu zamówienia lub w przypadku zamknięcia placówki, dostarcza zamagazynowane w niej części do innej posesji jeśli taka istnieje.
    - Ułatwia użytkownikowi wprowadzanie danych dotyczących zleceń, gwarancji, zamówień itd.
    - Pozwala w łatwy sposób podać informację o przeniesieniu części do naprawy z jednej placówki do drugiej.
    - Wyświetla informacje (w ujęciu miesięcznym) na temat pieniędzy zarobionych z zleceń i gwarancji, oraz kosztów poniesionych w ramach zamówień części z hurtowni.
    - Potrafi filtrować pracowników zatrudnionych od zwolnionych, oraz od tych będących na urlopie.
    - Umożliwia wyświetlanie jakie zlecenia są realizowane w danej chwili przez pracownika.
    - Pozostałe funkcjonalności są omawiane przy opisie implementacji.

- **Ograniczenia przyjęte przy projektowaniu**:
    - Baza danych przechowuje tylko faktyczną cenę jaką ponosi klient w ramach zlecenia/gwarancji.
    - Analogicznie z zamówieniami z hurtowni.
    - Przechowuje tylko aktualną (w sensie bieżącą) cenę pojedynczej części z Hurtowni.
    - Jeśli cena się zmieniła w czasie, to informację o niej można uzyskać jedynie z historii zamówień, gdzie przy danej części będzie widnieć inna cena niż przy aktualnej.
    - Każda część ma określonego dostawcę, tzn. nie można zakupić tej części z innej hurtowni jeśli nie jest ona zapisana jako dostawca tej części.
    - Zamówienia z jednej hurtowni odbywają się w ramach dnia i placówki, tzn. jeżeli zostanie złożone zamówienie na daną część, to jeżeli istniało już zamówienie z tej samej placówki z tego samego dnia, to składane zamówienie będzie dołączonego do już istniejącego.

Diagram

Poniższy diagram przedstawia wszystkie tabele w bazie danych, oraz ich powiązania między sobą.

Aby uprościć zrozumienie zależności między tabelami, przy opisie każdej z nich pojawi się mniejszy diagram pokazujący jej powiązania.



Tabele

Rabaty

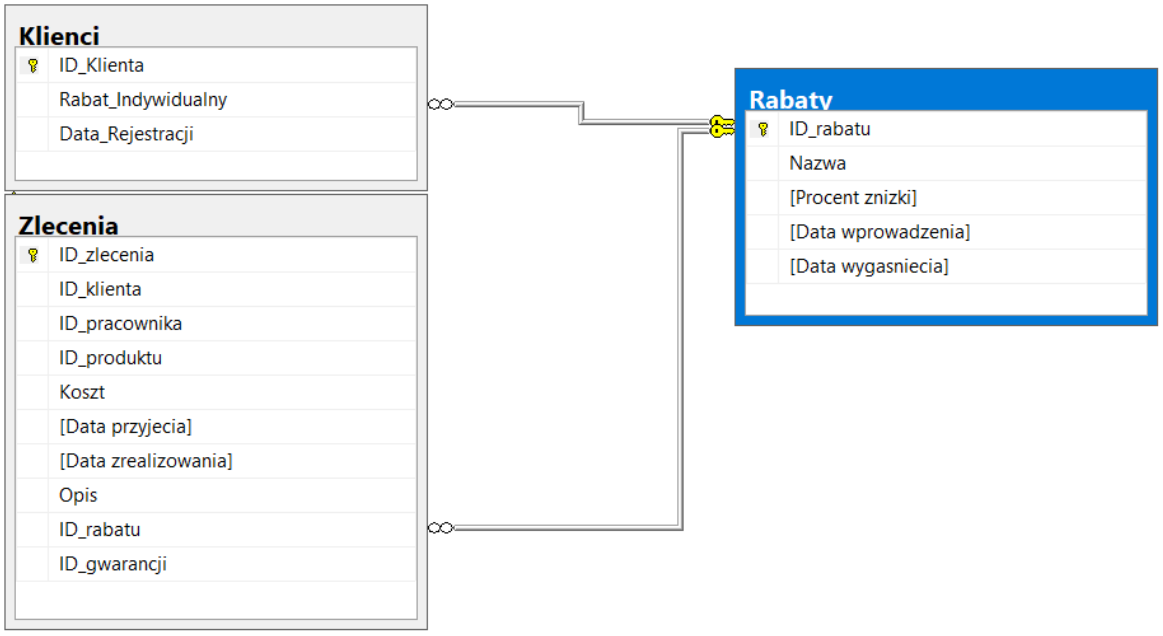
Wizualizacja tabeli

ID_rabatu	Nazwa	Procent zniżki	Data wprowadzenia	Data wygaśnięcia
INT PK IDENTITY(1,1)	NVARCHAR(50) NOT NULL UNIQUE	REAL NOT NULL	DATE NOT NULL	DATE

Kod

```
CREATE TABLE rabaty
  (
     id_rabatu            INT PRIMARY KEY IDENTITY(1, 1),
     nazwa                NVARCHAR(50) NOT NULL UNIQUE,
     [procent znizki]     REAL NOT NULL,
     [data wprowadzenia] DATE NOT NULL,
     [data wygasniecia]  DATE
  )
```

Diagram powiązań



Opis

- Rabaty służą do zmniejszenia klientom cen zleceń i mają określoną datę ważności.
- Rabat można przypisać do bezpośrednio do klienta, oraz do zlecenia.

Osoby

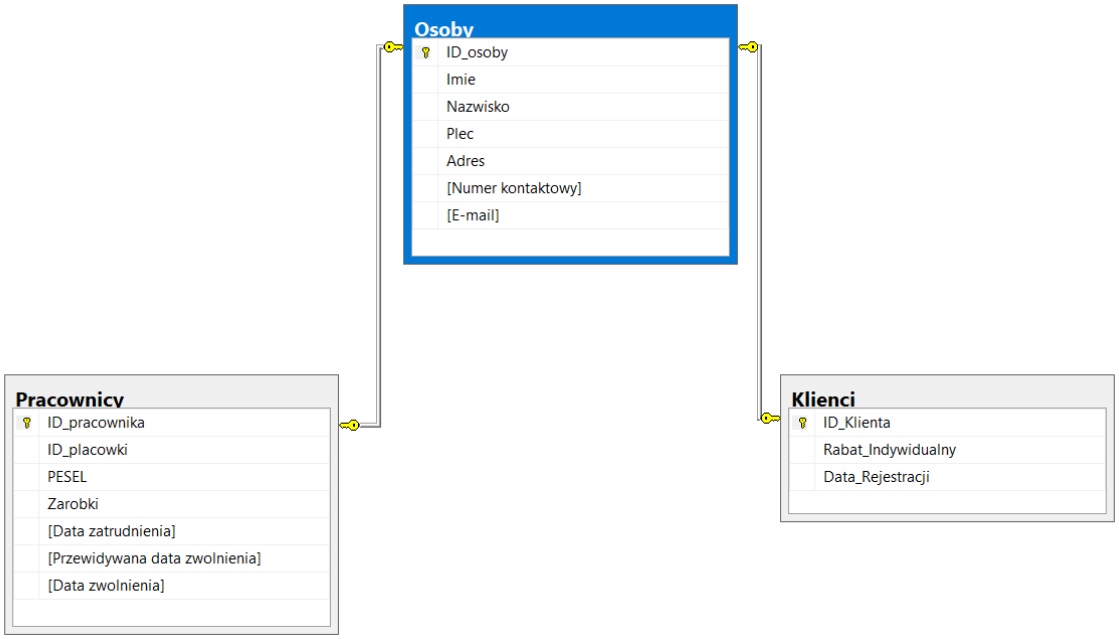
Wizualizacja tabeli

ID_osoby	Imie	Nazwisko	Plec	Adres	Numer kontaktowy	E-mail
INT PK IDENTITY(1,1)	NVARCHAR(50) NOT NULL	NVARCHAR(50) NOT NULL	CHAR(1) NOT NULL	VARCHAR(60)	VARCHAR(24)	NVRACHAR(60)

Kod

```
CREATE TABLE osoby
  (
     id_osoby           INT PRIMARY KEY IDENTITY(1, 1),
     imie               NVARCHAR(50) NOT NULL,
     nazwisko           NVARCHAR(50) NOT NULL,
     plec               CHAR(1) NOT NULL,
     adres              NVARCHAR(60),
     [numer kontaktowy] VARCHAR(24),
     [e-mail]           NVARCHAR(60),
     --Ograniczenie znaku płci
     CONSTRAINT [Znak plci] CHECK(plec IN ('M', 'K'))
  )
```

Diagram powiązań



Opis

- Tabela przedstawia podstawowe informacje o klientach oraz pracownikach.
- Na kolumnę **Plec** nałożony jest ogranicznik **CHECK** sprawdzający czy znak jest literą **K** lub **M**.

Placówki

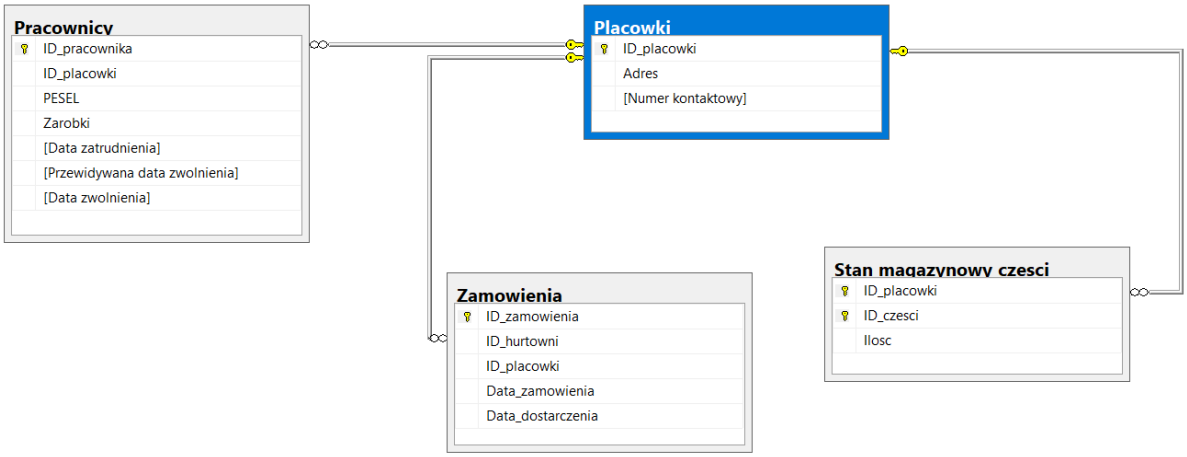
Wizualizacja tabeli

ID_placowki	Adres	Numer kontaktowy
INT	NVARCHAR(60)	VARCHAR(24)
PK	NOT NULL	
IDENTITY(1,1)	UNIQUE	

Kod

```
CREATE TABLE placowki
  (
     id_placowki         INT PRIMARY KEY IDENTITY(1, 1),
     adres               NVARCHAR(60) NOT NULL UNIQUE,
     [numer kontaktowy] VARCHAR(24)
  )
```

Diagram powiązań



Opis

- Serwis naprawczy może posiadać kilka placówek.
- Każda placówka zatrudnia swoich pracowników którzy realizują w niej zlecenia.
- Każda placówka jest dodatkowo magazynem na części, co symbolizuje dowiązanie

Pracownicy

Wizualizacja tabeli

ID_pracownika	ID_placowki	PESEL	Zarobki	Data zatrudnienia	Przewidywana data zwolnienia	Data zwolnienia
INT PK	INT NOT NULL	CHAR(11) NOT NULL UNIQUE	INT NOT NULL	DATE NOT NULL	DATE	DATE

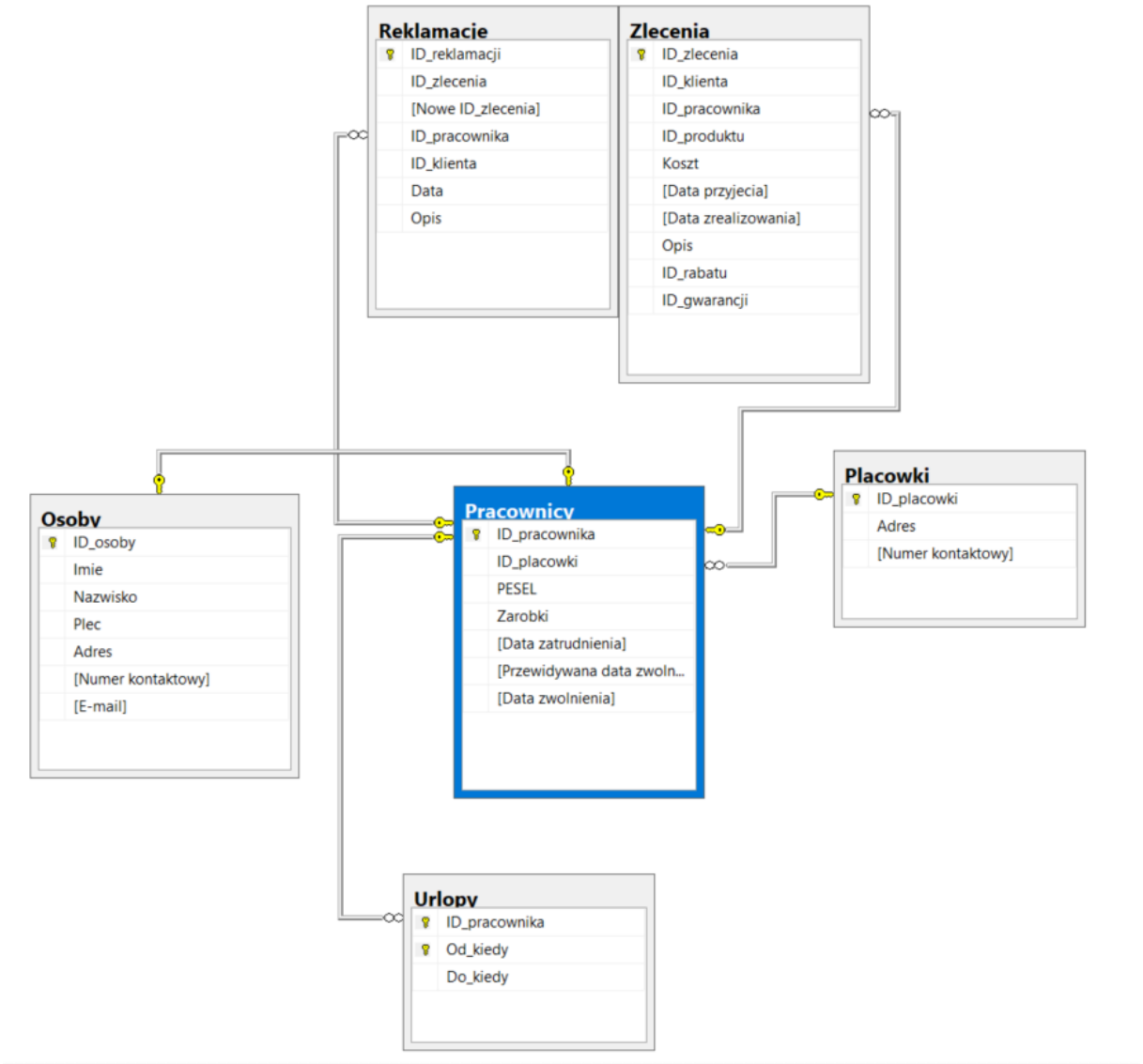
Kod

```
CREATE TABLE pracownicy
  (
     id_pracownika                  INT PRIMARY KEY,
     id_placowki                    INT NOT NULL,
     pesel                          CHAR(11) NOT NULL UNIQUE,
     zarobki                        INT NOT NULL CHECK(zarobki >= 0),
     [data zatrudnienia]            DATE NOT NULL,
     [przewidywana data zwolnienia] DATE,
     [data zwolnienia]              DATE,
     FOREIGN KEY (id_pracownika) REFERENCES osoby(id_osoby),
     FOREIGN KEY (id_placowki) REFERENCES placowki(id_placowki) ON DELETE
     CASCADE
  )
```

Opis

- Tak jak mówi nazwa tabeli, dostarcza ona informacji o pracownikach.
- Każdy pracownik jest przypisany do danej placówki.
- Jeżeli pracownik ma umowę na czas nieokreślony do pole **Przewidywana data zwolnienia** jest **Nullem**.
- Jeżeli **Data zwolnienia** jest **Nullem** to znaczy że pracownik nie został zwolniony i dalej pracuje w firmie.
- Zwolnieni pracownicy nie mogą wykonywać zleceń.
- Ta tabela dziedziczy po tabeli **Osoby**.

Diagram powiązań



Klienci

Wizualizacja tabeli

ID_klienta	Rabat_Indywidualny	Data_Rejestracji
INT	INT	DATE
PK		NOT NULL

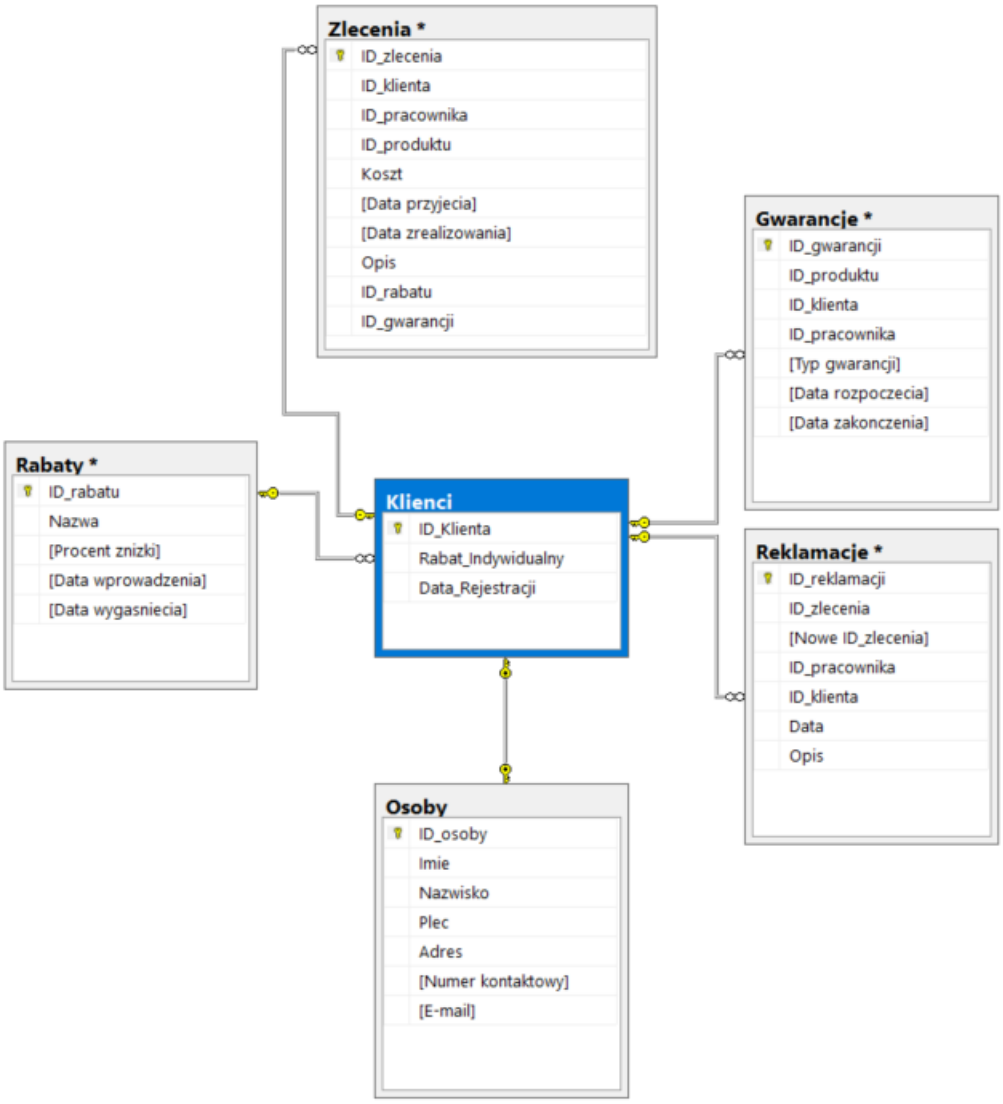
Kod

```
CREATE TABLE klienci
  (
     id_klienta         INT PRIMARY KEY,
     rabat_indywidualny INT,
     data_rejestracji   DATE NOT NULL,
     FOREIGN KEY(id_klienta) REFERENCES osoby(id_osoby) ON DELETE CASCADE,
     FOREIGN KEY(rabat_indywidualny) REFERENCES rabaty(id_rabatu) ON DELETE SET
     NULL
  )
```

Opis

- Tabel dziedziczy po tabeli **Osoby**.
- Każdemu klientowi można przypisać konkretny rabat.
- Dodatkowo przechowywana jest informacja kiedy klient został zapisany do bazy danych.
- Klient jest zleceniodawcą, oraz może zakupywać gwarancje.

Diagram powiązań



Urlopy

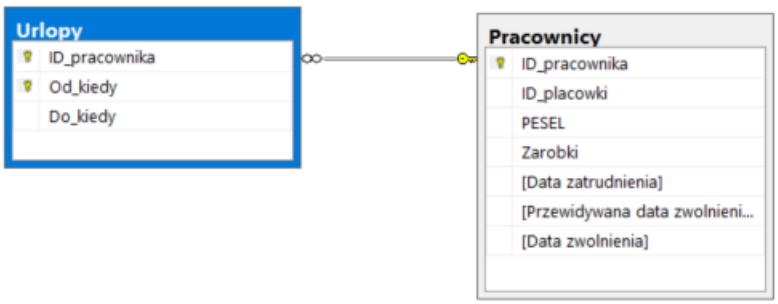
Wizualizacja tabeli

ID_pracownika	Od_kiedy	Do_kiedy
INT	DATE	DATE
PK	PK	NOT NULL
	NOT NULL	

Kod

```
CREATE TABLE urlopy
  (
     id_pracownika INT,
     od_kiedy      DATE NOT NULL,
     do_kiedy      DATE NOT NULL,
     PRIMARY KEY(id_pracownika, od_kiedy),
     FOREIGN KEY (id_pracownika) REFERENCES pracownicy(id_pracownika) ON DELETE
     CASCADE
  )
```

Diagram powiązań



Opis

- Każdy pracownik może zażądać urlopu.
- Klucz główny jest złożony z atrybutów **ID_pracownika**, oraz **Od_kiedy**.
- W przypadku usunięcia pracownika, wszystkie jego urlopy zostaną automatycznie usunięte.

Hurtownie

Wizualizacja tabeli

ID_hurtowni	Nazwa	Adres	Numer kontaktowy	Opis
INT PK	NVARCHAR(30) NOT NULL UNIQUE	NVARCHAR(60) NOT NULL UNIQUE	NVARCHAR(24)	NVARCHAR(255)

Kod

```
CREATE TABLE hurtownie
  (
     id_hurtowni         INT PRIMARY KEY IDENTITY(1, 1),
     nazwa               NVARCHAR(30) NOT NULL UNIQUE,
     adres               NVARCHAR(60) NOT NULL UNIQUE,
     [numer kontaktowy] NVARCHAR(24),
     [opis]              NVARCHAR(255)
  )
```

Diagram powiązań

Opis

- Hurtownia jest dostawcą części do placówek.
- Każda hurtownia ma swoją nazwę, oraz adres gdzie się znajduje.
- Opcjonalnie numer kontaktowy do niej, oraz opis co dokładnie dostarcza hurtownia.

Kategorie

Wizualizacja tabeli

ID_kategorii	Nazwa	Opis
INT	NVARCHAR(15)	NVARCHAR(255)
PK	NOT NULL	
IDENTITY(1,1)	UNIQUE	

Kod

```
CREATE TABLE kategorie
  (
     id_kategorii INT PRIMARY KEY IDENTITY(1, 1),
     nazwa        NVARCHAR(15) NOT NULL UNIQUE,
     opis         NVARCHAR(255)
  )
```

Diagram powiązań

Opis

- Każdy zlecony produkt do naprawy musi zostać podpięty do jakieś kategorii produktów z tej tabeli.
- Dodatkowo może być dostarczony opis.

Produkty do naprawy

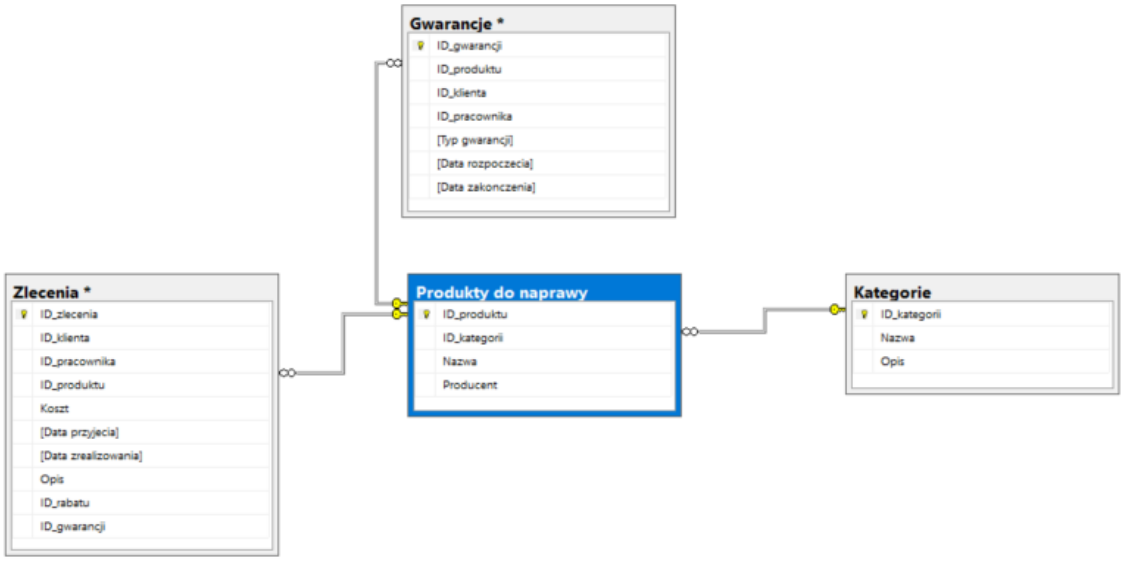
Wizualizacja tabeli

ID_produktu	ID_kategorii	Nazwa	Producent
INT	INT	NVARCHAR(15)	NVARCHAR(24)
PK	NOT NULL	NOT NULL	
IDENTITY(1,1)		UNIQUE	

Kod

```
CREATE TABLE [produkty do naprawy]
  (
     id_produktu  INT PRIMARY KEY IDENTITY(1, 1),
     id_kategorii INT NOT NULL,
     nazwa        NVARCHAR(15) NOT NULL UNIQUE,
     producent    NVARCHAR(24)
     FOREIGN KEY (id_kategorii) REFERENCES kategorie(id_kategorii)
  )
```

Diagram powiązań



Opis

- Tabela dostarcza informacji o produktach które klienci złożyli do naprawy.
- Każdy produkt należy do jakiejś kategorii i ma swoją nazwę, oraz opcjonalnie producenta.
- Jeżeli klient zdecyduje się na zakup gwarancji, to kupuje ją na dokładnie jeden produkt.

Czesci do naprawy

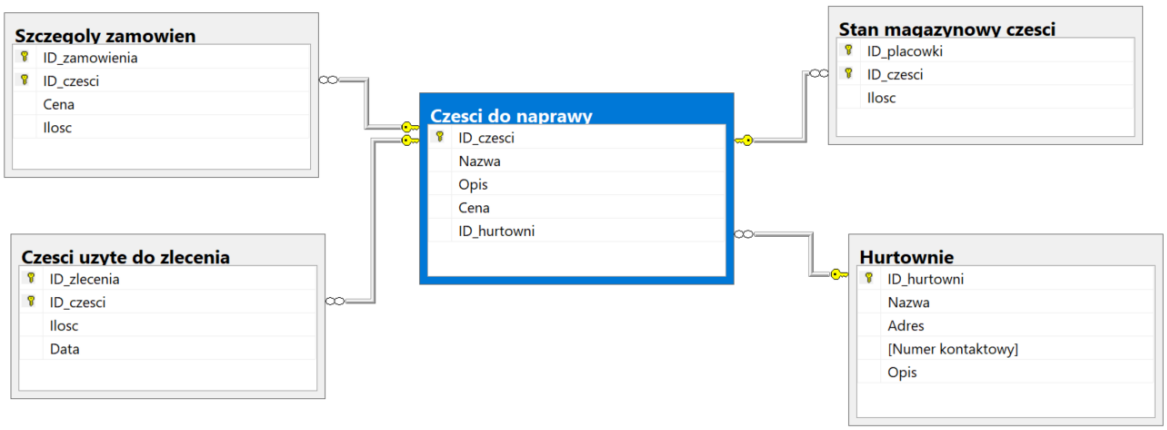
Wizualizacja tabeli

ID_czesci	Nazwa	Opis	Cena	ID_hurtowni
INT	NVARCHAR(15)	NVARCHAR(255)	MONEY	INT
PK	NOT NULL		NOT NULL	NOT NULL
IDENTITY(1,1)	UNIQUE			

Kod

```
CREATE TABLE [czesci do naprawy]
  (
     id_czesci    INT PRIMARY KEY IDENTITY(1, 1),
     nazwa        NVARCHAR(15) NOT NULL UNIQUE,
     opis         NVARCHAR(255),
     cena         MONEY NOT NULL,
     id_hurtowni INT NOT NULL,
     FOREIGN KEY (id_hurtowni) REFERENCES hurtownie(id_hurtowni)
  )
```

Diagram powiązań



Opis

- Każdy wiersz tabeli dostarcza informację o konkretnej części która może być wymagana do naprawienia produktu przez pracownika.
- Ma on swoją nazwę i opcjonalnie opis.
- Atrybut **Cena** dostarcza informacji ile kosztuje pojedyncza część, a **ID_hurtowni** kto jest dostawcą tej części.

Stan magazynowy części

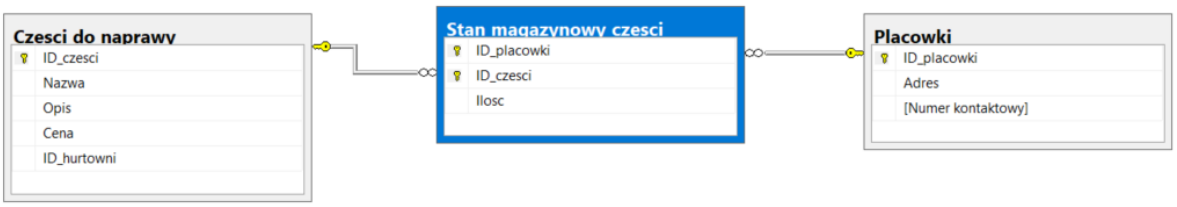
Wizualizacja tabeli

ID_placowki	ID_czesci	Ilosc
INT	INT	INT
PK	PK	NOT NULL

Kod

```
CREATE TABLE [stan magazynowy czesci]
  (
     id_placowki INT,
     id_czesci   INT,
     ilosc       INT NOT NULL CHECK(ilosc >= 0),
     PRIMARY KEY(id_placowki, id_czesci),
     FOREIGN KEY (id_placowki) REFERENCES placowki(id_placowki),
     FOREIGN KEY (id_czesci) REFERENCES [czesci do naprawy](id_czesci),
  )
```

Diagram powiązań



Opis

- Klucz główny w tabeli jest złożony z **ID_placowki** i **ID_czesci**.
- Ilość części w placówce jest zmienna w czasie i kupuje się je do placówki za pomocą zamówień do hurtowni.
- Na Ilość towarów w magazynie nałożony jest warunek **CHECK >= 0**.

Zamowienia

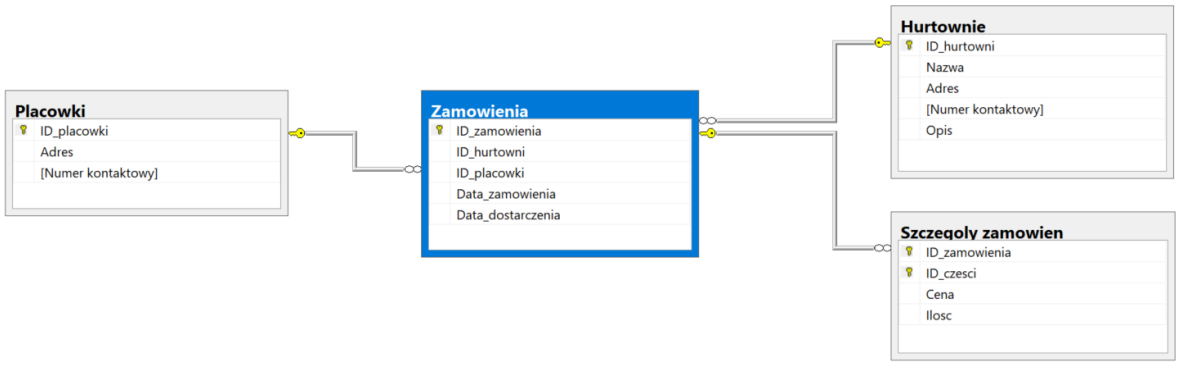
Wizualizacja tabeli

ID_zamowienia	ID_hurtowni	ID_placowki	Data_zamowienia	Data_dostarczenia
INT	INT	INT	DATE	DATE
PK	NOT NULL	NOT NULL	NOT NULL	
IDENTITY(1,1)				

Kod

```
CREATE TABLE zamowienia
  (
     id_zamowienia     INT PRIMARY KEY IDENTITY(1, 1),
     id_hurtowni       INT NOT NULL,
     id_placowki       INT NOT NULL,
     data_zamowienia   DATE NOT NULL,
     data_dostarczenia DATE,
     FOREIGN KEY (id_hurtowni) REFERENCES hurtownie(id_hurtowni),
     FOREIGN KEY (id_placowki) REFERENCES placowki(id_placowki)
  )
```

Diagram powiązań



Opis

- Tabela ta przedstawia zamówienia części z hurtowni który jest dostawcom zdefiniowanym w tabeli **Czesci do naprawy**.
- Każdego dnia w ramach jednej placówki i jednej hurtowni, istnieje tylko jeden rekord z zamówieniem.
- Jeżeli **Data_dostarczenia** jest **Nullem** to znaczy że zamówienie nie dotarło jeszcze do placówki.
- Jeżeli w pole **Data_dostarczenia** zostanie wpisana data, to zamawiane części (zdefiniowane w tabeli **Szczegoly zamowien**) będą przesłane do placówki.

Szczegoly zamowien

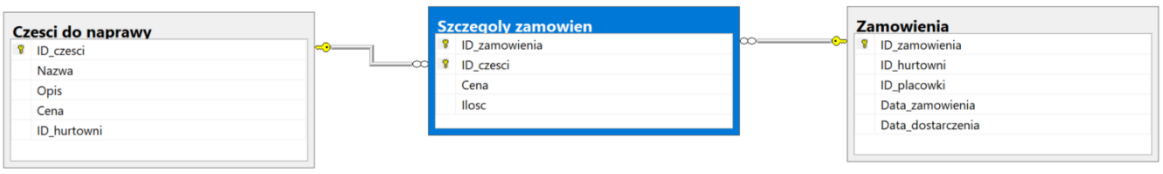
Wizualizacja tabeli

ID_zamowienia	ID_czesci	Cena	Ilosc
INT	INT	MONEY	INT
PK	PK	NOT NULL	NOT NULL

Kod

```
CREATE TABLE [szczegoly zamowien]
  (
     id_zamowienia INT,
     id_czesci     INT,
     cena          MONEY NOT NULL,
     ilosc         INT NOT NULL CHECK(ilosc >= 1),
     PRIMARY KEY (id_zamowienia, id_czesci),
     FOREIGN KEY (id_zamowienia) REFERENCES [zamowienia](id_zamowienia) ON
     DELETE CASCADE,
     FOREIGN KEY (id_czesci) REFERENCES [czesci do naprawy](id_czesci)
  )
```

Diagram powiązań



Opis

- Kluczem główny jest złożony. Składa się z pół **ID_zamowienia** i **ID_czesci**.
- Pole **Cena** dostarcza informacji o tym ile kosztowały wszystkie części w momencie realizacji tego zamówienia.
- Na pole **Ilosc** nałożony jest warunek **CHECK >= 1**.

Typ gwarancji

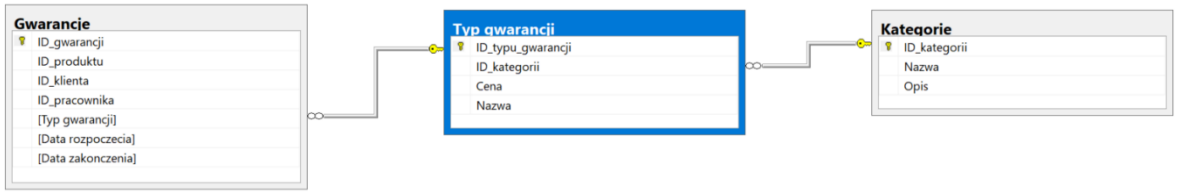
Wizualizacja tabeli

ID_typu_gwarancji	ID_kategorii	Cena	Czas trwania	Nazwa
INT PK IDENTITY(1,1)	INT NOT NULL	MONEY NOT NULL	INT NOT NULL	NVARCHAR(50) NOT NULL UNIQUE

Kod

```
CREATE TABLE [typ gwarancji]
  (
     id_typu_gwarancji INT PRIMARY KEY IDENTITY(1, 1),
     id_kategorii      INT NOT NULL,
     cena              MONEY NOT NULL,
     [czas trwania]    INT NOT NULL,
     nazwa             NVARCHAR(50) NOT NULL UNIQUE,
     FOREIGN KEY (id_kategorii) REFERENCES kategorie(id_kategorii) ON DELETE
     CASCADE
  )
```

Diagram powiązań



Opis

- Definiuje rodzaj gwarancji jaką może zakupić klient.
- Każda gwarancja jest przypisana do jakiejś kategorii i ma swoją cenę oraz nazwę.

Gwarancje

Wizualizacja tabeli

ID_gwarancji	ID_produktu	ID_klienta	ID_pracownika	Typ gwarancji	Data rozpoczecia	Data zakonczenia
INT PK IDENTITY(1,1)	INT NOT NULL	INT NOT NULL	INT	DATE NOT NULL	DATE NOT NULL	DATE NOT NULL

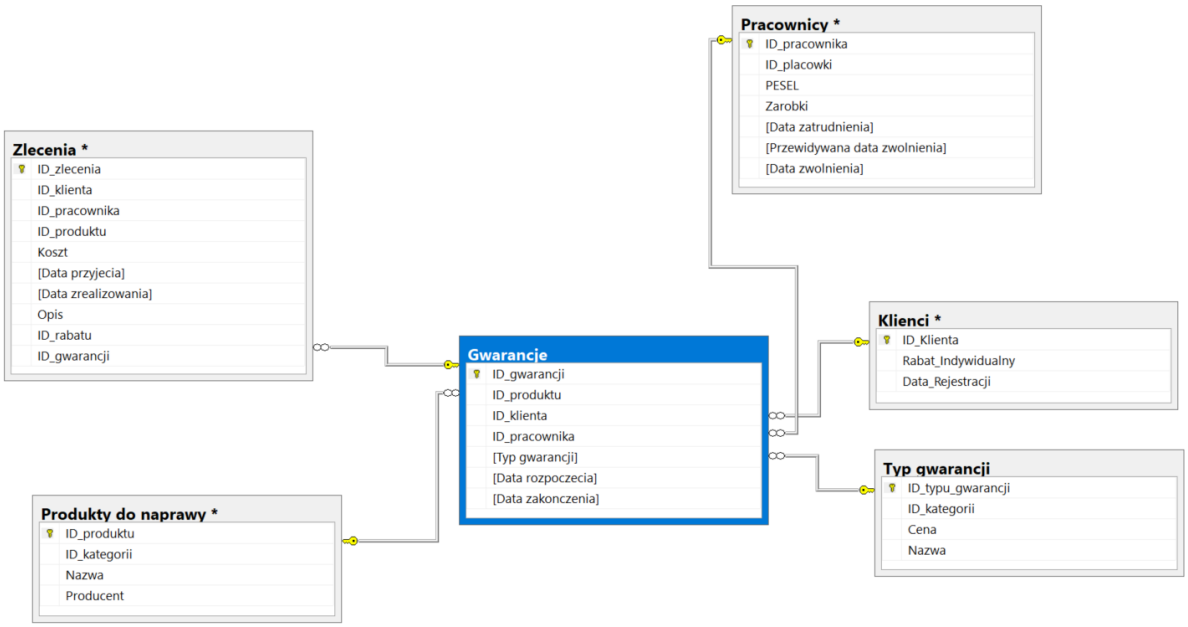
Kod

```
CREATE TABLE gwarancje
  (
     id_gwarancji       INT PRIMARY KEY IDENTITY(1, 1),
     id_produktu        INT NOT NULL,
     id_klienta         INT NOT NULL,
     id_pracownika      INT,
     [typ gwarancji]    INT NOT NULL,
     [data rozpoczecia] DATE NOT NULL,
     [data zakonczenia] DATE NOT NULL,
     FOREIGN KEY (id_klienta) REFERENCES klienci(id_klienta),
     FOREIGN KEY (id_produktu) REFERENCES [produkty do naprawy](id_produktu),
     FOREIGN KEY (id_pracownika) REFERENCES pracownicy(id_pracownika) ON DELETE
     SET NULL,
     FOREIGN KEY ([typ gwarancji]) REFERENCES [typ gwarancji](id_typu_gwarancji)
  )
```

Opis

- Gwarancję może zakupić klient reprezentowany przez **ID_klienta**, na konkretny produkt (**ID_produktu**).
- **ID_pracownika** oznacza pracownika który dokonał sprzedaży usługi gwarancyjnej klientowi, jeżeli pracownik zostanie usunięty z Bazy Danych to zostanie usunięty z każdego rekordu i zamieniony na **NULL**.
- **Typ gwarancji** oznacza typ gwarancji jakim został objęty produkt, oraz jej koszt.
- Kategoria **ID_produktu** musi zgadzać się z kategorią na jakiej zdefiniowany jest **Typ gwarancji**.
- **Data rozpoczecia** i **Data zakonczenia** wyznacza okres na jaki działa usługa.
- Zakupienie przez klienta gwarancji, pozwala mu zgłaszać produkt do naprawy ze zerową cenę.

Diagram powiązań



Zlecenia

Wizualizacja tabeli

Poniższa wizualizacja została podzielona na dwie części, ponieważ była za duża.

ID_zlecenia	ID_klienta	ID_pracownika	ID_produktu	Koszt
INT	INT	INT	INT	MONEY
PK	NOT NULL	NOT NULL	NOT NULL	NOT NULL
IDENTITY(1,1)				

Data przyjęcia	Data zrealizowania	Opis	ID_rabatu	ID_gwarancji
DATE	DATE	NVARCHAR(255)	INT	INT
NOT NULL				

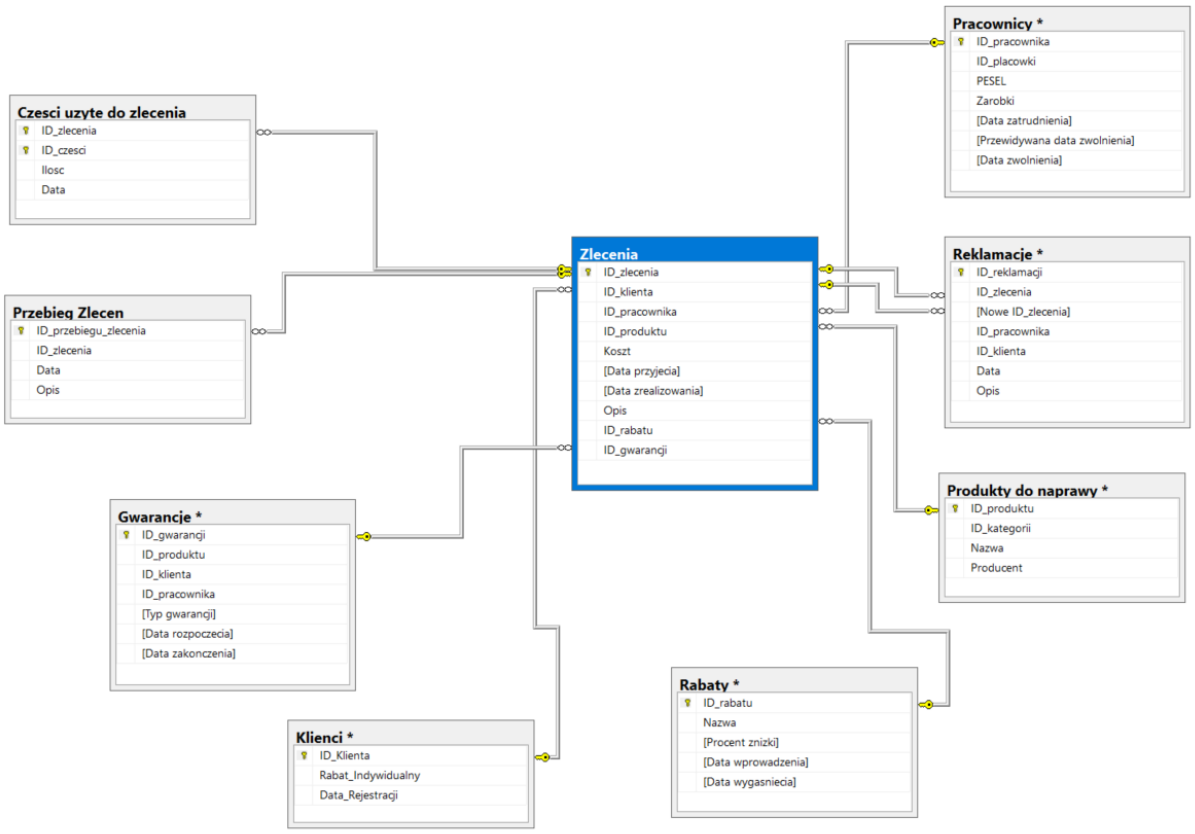
Kod

```
CREATE TABLE zlecenia
  (
     id_zlecenia          INT PRIMARY KEY IDENTITY(1, 1),
     id_klienta           INT NOT NULL,
     id_pracownika        INT NOT NULL,
     id_produktu          INT NOT NULL,
     koszt                MONEY NOT NULL,
     [data przyjecia]     DATE NOT NULL,
     [data zrealizowania] DATE,
     opis                 NVARCHAR(255),
     id_rabatu            INT,
     id_gwarancji         INT,
     FOREIGN KEY (id_klienta) REFERENCES klienci(id_klienta),
     FOREIGN KEY (id_pracownika) REFERENCES pracownicy(id_pracownika),
     FOREIGN KEY (id_produktu) REFERENCES [produkty do naprawy](id_produktu),
     FOREIGN KEY (id_gwarancji) REFERENCES gwarancje(id_gwarancji),
     FOREIGN KEY (id_rabatu) REFERENCES rabaty(id_rabatu)
  )
```

Opis

- Każde zlecenie jest składane przez klienta (**ID_klienta**), a realizowane przez pracownika (**ID_pracownika**).
- ID_produktu wskazuje jaki produkt został zlecony do naprawy.
- Koszt to cena brutto zlecenia.
    - Obliczana jest za pomocą pierwotnej ceny – procent rabatu na tym zleceniu – procent rabatu indywidualnego klienta.
    - Jeżeli klient posiada gwarancję to koszt zlecenia jest zerowy.
- Przy każdym zleceniu można podać rabat który obniża jego koszt.
- Pole **ID_gwarancji** wskazuje na to czy klient ma gwarancję na ten produkt. Jeżeli nie miał to pole jest **Nullem**.

Diagram powiązań



Przebieg Zlecen

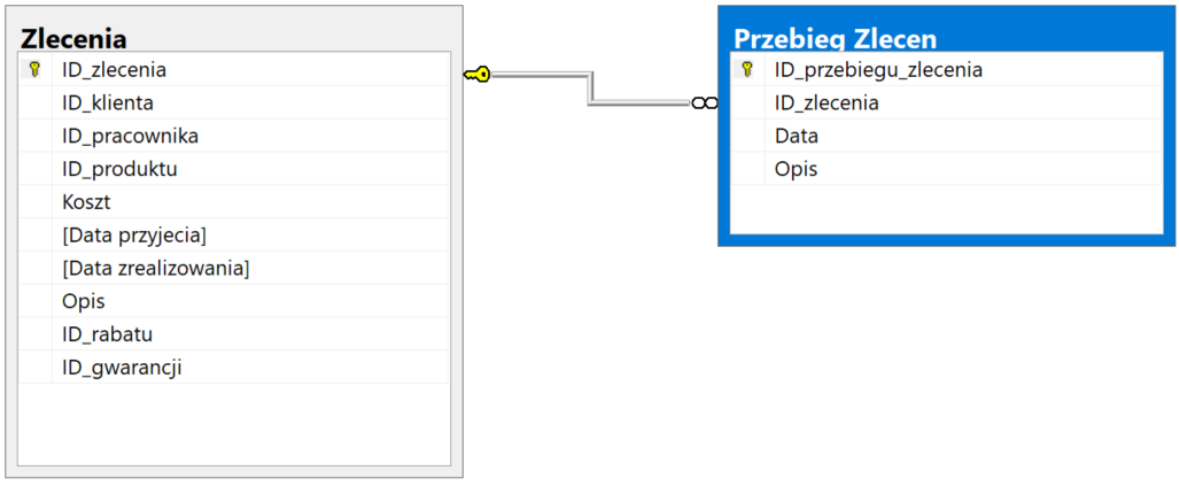
Wizualizacja tabeli

ID_przebiegu_zlecenia	ID_zlecenia	Data	Opis
INT	INT	DATE	NVARCHAR(255)
PK	NOT NULL	NOT NULL	NOT NULL

Kod

```
CREATE TABLE [przebieg zlecen]
  (
     id_przebiegu_zlecenia INT PRIMARY KEY IDENTITY(1, 1),
     id_zlecenia           INT NOT NULL,
     data                  DATE NOT NULL,
     opis                  NVARCHAR(255) NOT NULL,
     FOREIGN KEY (id_zlecenia) REFERENCES zlecenia(id_zlecenia) ON DELETE
     CASCADE
  )
```

Diagram powiązań



Opis

- Tabela zawierająca informację o przebiegu zleceń.
- Zlecenie definiowane jest przez **ID_zlecenia**, a kolejność akcji wykonanych w ramach zlecenia przez **Data**.
- Wymagany jest **Opis** co zostało wykonane w ramach zlecenia.

Czesci uzyte do zlecenia

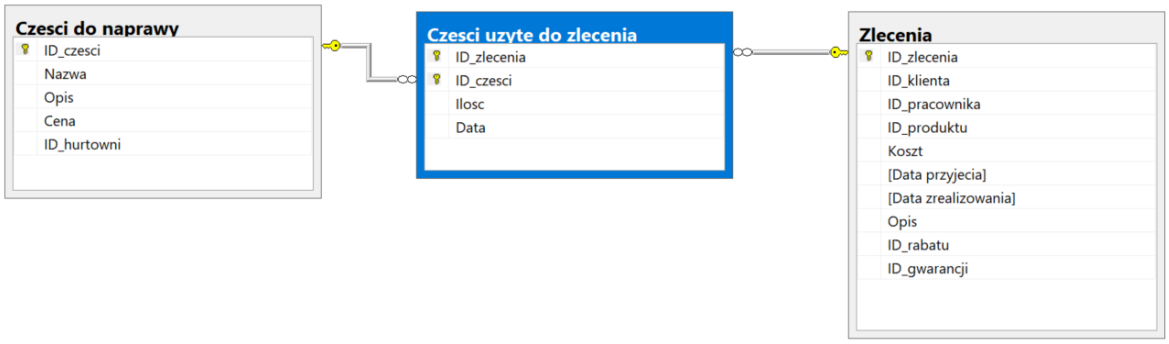
Wizualizacja tabeli

ID_zlecenia	ID_czesci	Ilosc	Data
INT	INT	INT	DATE
PK	PK	NOT NULL	NOT NULL

Kod

```
CREATE TABLE [czesci uzyte do zlecenia]
  (
     id_zlecenia INT,
     id_czesci   INT,
     ilosc       INT NOT NULL CHECK(ilosc >= 1),
     data        DATE NOT NULL,
     PRIMARY KEY(id_zlecenia, id_czesci),
     FOREIGN KEY (id_zlecenia) REFERENCES zlecenia(id_zlecenia) ON DELETE
     CASCADE,
     FOREIGN KEY (id_czesci) REFERENCES [czesci do naprawy](id_czesci),
  )
```

Diagram powiązań



Opis

- W ramach zlecenia potrzebne są części które potrzebuje naprawiający.
- Części które pobierane są z magazynu w ramach zlecenia, są zapisywane w tej tabeli.
- Na pole **Ilosc** nałożony jest warunek **CHECK >= 1**.
- W przypadku usunięcia zlecenia z Bazy Danych, cała historia użytych części zostanie usunięta.

Reklamacje

Wizualizacja tabeli

ID_reklamacji	ID_zlecenia	Nowe ID_zlecenia	ID_pracownika	ID_klienta	DATA	Opis
INT PK IDENTITY(1,1)	INT NOT NULL	INT NOT NULL	INT	INT NOT NULL	DATE NOT NULL	NVARCHAR(255)

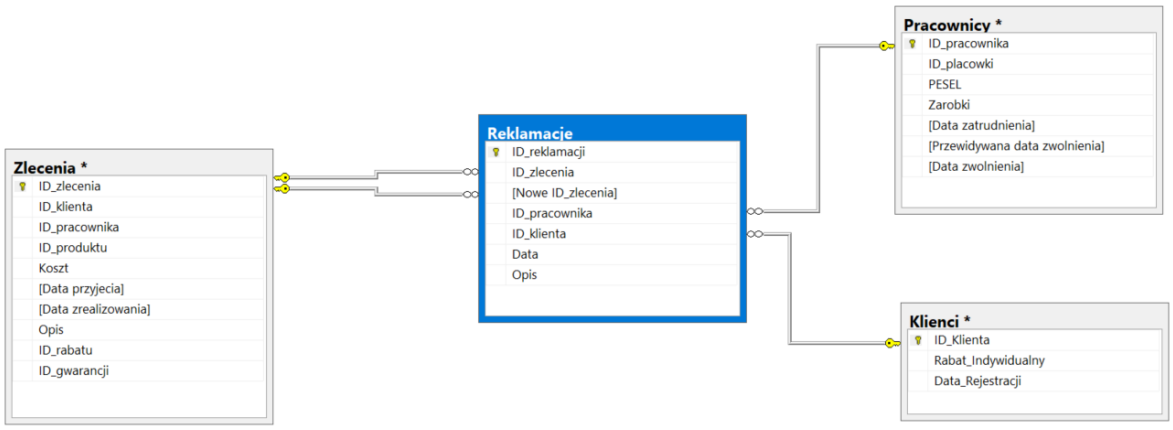
Kod

```
CREATE TABLE reklamacje
  (
     id_reklamacji      INT PRIMARY KEY IDENTITY(1, 1),
     id_zlecenia        INT NOT NULL,
     [nowe id_zlecenia] INT NOT NULL,
     id_pracownika      INT,
     id_klienta         INT NOT NULL,
     data               DATE NOT NULL,
     opis               NVARCHAR(255),
     FOREIGN KEY (id_klienta) REFERENCES klienci(id_klienta),
     FOREIGN KEY (id_zlecenia) REFERENCES zlecenia(id_zlecenia),
     FOREIGN KEY ([nowe id_zlecenia]) REFERENCES zlecenia(id_zlecenia),
     FOREIGN KEY (id_pracownika) REFERENCES pracownicy(id_pracownika) ON DELETE
     SET NULL
  )
```

Opis

- Jeżeli klient uzna że zlecenie zostało źle wykonane i produkt nie został naprawiony, to ma prawo do reklamacji.
- **ID_zlecenia** wskazuje na zlecenie które zostało niepoprawnie wykonane.
- **Nowe ID_zlecenia** wskazuje na nowe zlecenie w ramach którego rozpatrywana jest ta reklamacja.
- **ID_pracownika** wskazuje tutaj na pracownika który przyjął reklamacje (niekoniecznie się nią zajmuje), jeżeli zostanie on usunięty z Bazy Danych to w to miejsce pojawi się **NULL.**

Diagram powiązań



Funkcje i Widoki

Informacje_o_zamowieniu

Typ:

Funkcja

Argumenty:

@ID_zamowienia INT

Typ zwracany:

ID_zamowienia	ID_hurtowni	ID_placowki	Cena	Data_zamowienia	Data_dostarczenia
INT	INT	INT	MONEY	DATE	DATE

Opis:

- Funkcja dostarcza informacji o konkretnym zamówieniu, reprezentowanym przez argument **@ID_zamowienia**.
- **Cena** jest sumaryczną ceną wszystkich zamawianych części w obrębie danego zamówienia.

Kod:

```
CREATE FUNCTION Informacje_o_zamowieniu (@ID_Zamowienia INT)
RETURNS TABLE
AS
RETURN (
            SELECT Z.ID_zamowienia, Z.ID_hurtowni, Z.ID_placowki, SUM(CAST(S.Cena AS
MONEY)) AS Koszt, Z.Data_zamowienia, Z.Data_dostarczenia
                  FROM Zamowienia AS Z
                  JOIN [Szczegoly zamowien] AS S ON (Z.ID_zamowienia =
S.ID_zamowienia)
            WHERE Z.ID_zamowienia = @ID_Zamowienia
            GROUP BY Z.ID_zamowienia, Z.ID_hurtowni, Z.ID_placowki,
Z.Data_zamowienia, Z.Data_dostarczenia
)
```

Pracownicy_na_urlopie

Typ:

Widok

Typ zwracany:

ID_pracownika	ID_placowki	Od_kiedy	Do_kiedy
INT	INT	DATE	DATE

Opis:

- Widok reprezentuje wszystkich zatrudnionych pracowników, którzy są obecnie na urlopie.

Kod:

```
CREATE VIEW Pracownicy_na_urlopie
AS
	SELECT U.ID_pracownika, P.ID_placowki, U.Od_kiedy, U.Do_kiedy
		FROM Urlopy AS U
		JOIN Pracownicy AS P ON (P.ID_pracownika = U.ID_pracownika)
	WHERE Do_kiedy > GETDATE() --Przed końcem urlopu
	AND P.[Data zwolnienia] IS NULL --Pracownik który nie został zwolniony
```

Aktualnie_zatrudnieni_pracownicy

Typ:

Widok

Typ zwracany:

ID_pracownika	ID_placowki	PESEL	Data zatrudnienia	Przewidywana data zwolnienia
INT	INT	CHAR(11)	DATE	DATE

Opis:

- Widok wyświetla wszystkich zatrudnionych pracowników we wszystkich placówkach.

Kod:

```
CREATE VIEW Aktualnie_zatrudnieni_pracownicy
AS
	SELECT P.ID_pracownika, P.ID_placowki, P.PESEL, P.[Data zatrudnienia],
P.[Przewidywana data zwolnienia]
		FROM Pracownicy AS P
		JOIN Osoby AS O ON (O.ID_osoby = P.ID_pracownika)
		WHERE [Data zwolnienia] IS NULL
```

Gwarancje_klienta

Typ:

Funkcja

Argumenty:

@ID_klienta INT.

Typ zwracany:

Tabela z kolumnami identycznymi jak w tabeli **Gwarancje**.

Opis:

- Funkcja zwraca informację o wszystkich aktywnych gwarancjach posiadanych przez pracownika.

Kod:

```
RETURNS TABLE
AS
RETURN (
            SELECT G.*
            FROM Gwarancje AS G
            WHERE ID_klienta = @ID_Klienta
            AND G.[Data zakonczenia] > GETDATE()
)
```

Zlecenia_dla_kategorii

Typ:

Funkcja

Argumenty:

@ID_kategorii INT

Typ zwracany:

Tabela z kolumnami identycznymi jak w tabeli **Zlecenia**.

Opis:

- Funkcja zwraca wszystkie wykonywane zlecenia, dla produktów kategorii danej argumentem **@ID_kategorii**.

Kod:

```
CREATE FUNCTION Zlecenia_dla_kategorii(@ID_kategorii INT)
RETURNS TABLE
AS
RETURN (
            SELECT Z.*
                  FROM Zlecenia AS Z
                  JOIN [Produkty do naprawy] AS P ON (P.ID_produktu = Z.ID_produktu)
            WHERE P.ID_kategorii = @ID_kategorii
            AND Z.[Data zrealizowania] IS NULL
)
```

Czesci_ze_wszystkich_placowek

Typ:

Widok

Typ zwracany:

ID_zamowienia	Ilosc
INT	INT

Opis:

- Widok zwraca ilość wszystkich części z placówek, z podziałem na konkretną część.

Kod:

```
CREATE VIEW Czesci_ze_wszystkich_placowek
AS
	SELECT S.ID_czesci, COUNT(S.Ilosc) [Ilosc]
		FROM [Stan magazynowy czesci] AS S
	GROUP BY S.ID_czesci
```

Aktualne_rabaty

Typ:

Widok

Typ zwracany:

Tabela takiej samej postaci jak tabela **Rabaty**.

Opis:

- Widok zwraca wszystkie rabaty, których data wygaśnięcia jeszcze nie minęła.
- Wszystkie rabaty wyświetlane przez ten widok, można przypisać klientowi lub do zlecenia.

Kod:

```
CREATE VIEW Aktualne_rabaty
AS
	SELECT *
		FROM Rabaty
	WHERE [Data wygasniecia] IS NULL
	OR [Data wygasniecia] < GETDATE()
```

Przychody_miesieczne

Typ:

Widok

Typ zwracany:

Data	Przychod ze zlecen	Przychod z gwarancji	Suma przychodow
VARCHAR(7)	MONEY	MONEY	MONEY

Opis:

- Widok zwraca zsumowane przychody ze zleceń, oraz z gwarancji w poszczególnym miesiącu, na podstawie daty przyjęcia zlecenia i zakupu gwarancji.
- Data ma postać **MM-YYYY** lub **M-YYYY** w zależności od długości zapisu miesiąca.

Kod:

```
CREATE VIEW Przychody_miesieczne
AS
	SELECT
		R.Data AS Data,
		SUM(CAST(R.[Przychod ze zlecen] AS MONEY)) AS [Przychod ze zlecen],
		SUM(CAST(R.[Przychod z gwarancji] AS MONEY)) AS [Przychod z gwarancji],
		SUM(CAST(R.[Przychod ze zlecen] + R.[Przychod z gwarancji] AS MONEY)) AS
[Suma przychodow]
	FROM
		(SELECT
			CAST(MONTH(Z.[Data przyjecia]) AS VARCHAR(2)) + '-' +
			CAST(YEAR(Z.[Data przyjecia]) AS VARCHAR(4)) AS Data,
			Z.Koszt AS [Przychod ze zlecen],
			0.00 AS [Przychod z gwarancji]
		FROM Zlecenia AS Z
		UNION ALL
		SELECT
			CAST(MONTH(G.[Data rozpoczecia]) AS VARCHAR(2)) + '-' +
			CAST(YEAR(G.[Data rozpoczecia]) AS VARCHAR(4)) AS Data,
			0.00 AS [Przychod ze zlecen],
			T.Cena AS [Przychod z gwarancji]
		FROM Gwarancje AS G
		JOIN [Typ gwarancji] T ON (G.[Typ gwarancji] = T.ID_typu_gwarancji)
		) AS R
	GROUP BY Data
```

Calkowity_koszt_zamówienia

Typ:

Funkcja

Argumenty:

@ID_zamowienia INT

Typ zwracany:

INT

Opis:

- Funkcja zwraca sumaryczny koszt zamówienia podanego argumentem **@ID_zamowienia**.

Kod:

```
CREATE FUNCTION Calkowity_koszt_zamowienia(@ID_zamowienia INT)
RETURNS INT
AS
BEGIN
RETURN (
	SELECT SUM(S.Cena)
		FROM [Szczegoly zamowien] AS S
	WHERE S.ID_zamowienia = @ID_zamowienia
)
END
```

Wydatki_miesieczne

Typ:

Widok

Typ zwracany

Data	Suma wydatkow
VARCHAR(7)	MONEY

Opis:

- Widok wyświetla zsumowane wydatki na zamówieniach z danego miesiące (na podstawie daty zrealizowania).
- Data ma postać **MM-YYYY** lub **M-YYYY** w zależności od długości zapisu miesiąca.

Kod:

```
CREATE VIEW Wydatki_miesieczne
AS
	SELECT
		CAST(MONTH(Z.Data_dostarczenia) AS VARCHAR(2)) + '-' +
		CAST(YEAR(Z.Data_dostarczenia)  AS VARCHAR(4)) AS Data,
		SUM(CAST(dbo.Calkowity_koszt_zamowienia(Z.ID_zamowienia) AS MONEY)) AS
		[Suma wydatkow]
	FROM Zamowienia AS Z
	GROUP BY CAST(MONTH(Z.Data_dostarczenia) AS VARCHAR(2)) + '-' +
CAST(YEAR(Z.Data_dostarczenia) AS VARCHAR(4))
```

Niezrealizowane_zlecenia

Typ:

Widok

Typ zwracany:

Tabela takiej samej postaci jak tabela **Zlecenia**.

Opis:

- Widok zwraca listę wszystkich zleceń które nie zostały jeszcze ukończone. Ich data zrealizowania jest **Nullem**.

Kod:

```
CREATE VIEW Niezrealizowane_zlecenia
AS
	SELECT *
		FROM Zlecenia AS Z
	WHERE Z.[Data zrealizowania] IS NULL
```

Przebieg_zlecenia

Typ:

Funkcja

Argumenty:

@ID_zlecenia INT

Typ zwracany:

ID_przebiegu_zlecenia	Data	Opis
INT	DATE	VARCHAR(255)

Opis:

- Funkcja zwraca przebieg zlecenia podanego argumentem **@ID_zlecenia**.
- Historia zmian w obrębie zlecenia wyświetlana jest od najnowszej, do najstarszej.
- Wyświetlanych jest tylko 100 najnowszych rekordów.

Kod:

```
CREATE FUNCTION Przebieg_zlecenia(@ID_zlecenia INT)
RETURNS TABLE
AS
RETURN (
	SELECT TOP 100 WITH TIES --Musi być TOP ponieważ nie da się dodać bez tego ORDER
BY w funkcji :(
		P.ID_przebiegu_zlecenia,
		P.Data,
		P.Opis
	FROM [Przebieg Zlecen] AS P
	ORDER BY P.Data DESC
)
```

Zlecenia_realizowane_przez_pracownika

Typ:

Funkcja

Argumenty:

@ID_pracownika INT

Typ zwracany:

ID_zlecenia	ID_klienta	Data przyjęcia	ID_produktu	Koszt	ID_gwarancji	ID_rabatu
INT	INT	DATE	INT	MONEY	INT	INT

Opis:

- Funkcja zwraca wszystkie zlecenia które obecnie realizuje pracownik.

Kod:

```
CREATE FUNCTION Zlecenia_realizowane_przez_pracownika(@ID_Pracownika INT)
RETURNS TABLE
AS
RETURN (
	SELECT Z.ID_zlecenia, Z.ID_klienta, Z.[Data przyjecia], Z.ID_produktu, Z.Koszt,
Z.ID_gwarancji, Z.ID_rabatu
		FROM Zlecenia AS Z
	WHERE Z.ID_pracownika = @ID_Pracownika
	AND Z.[Data zrealizowania] IS NULL
)
```

Pracownicy_zarabiajacy_mniej_niz

Typ:

Funkcja

Argumenty:

@Kwota INT

Typ zwracany:

Tabela taka sama jak tabela **Pracownicy**.

Opis:

- Zwraca wszystkich pracowników którzy zarabiają mniej niż kwota podana argumentem

Kod:

```
CREATE FUNCTION Pracownicy_zarabiajacy_mniej_niz(@Kwota INT)
RETURNS TABLE
AS
RETURN (
	SELECT *
		FROM Pracownicy
	WHERE Zarobki < @Kwota
)
```

Znajdz_produkt

Typ:

Funkcja

Argumenty

- **@nazwa_produktu NVARCHAR(15)**
- **@producent NVARCHAR(24)**
- **@kategoria NVARCHAR(15)**

Typ zwracany:

INT

Opis:

- Funkcja zwraca **ID_produktu** podanego definiowanego argumentami.
- Jeśli produktu nie ma w bazie danych, to zwracany jest **NULL**.

Kod:

```
CREATE FUNCTION Znajdz_produkt(@nazwa_produktu nvarchar(15), @producent nvarchar(24),
@kategoria nvarchar(15))
RETURNS INT
AS
BEGIN

	DECLARE @ID INT

	SELECT TOP 1 @ID=P.ID_produktu
	FROM [Produkty do naprawy] AS P RIGHT JOIN Kategorie AS K
	ON K.ID_kategorii = P.ID_kategorii
	WHERE @kategoria = K.Nazwa AND @nazwa_produktu = P.Nazwa AND @producent =
P.Producent

	RETURN @ID
END
```

Procedury

Zysk_z_dnia

- Argumenty:
  - **@Dzien DATE**
- Opis:
  - Procedura wyświetla zsumowany zysk ze zleceń, z konkretnego dnia podanego argumentem.

Nowy_produkt

- Argumenty:
  - **@nazwa_produktu NVARCHAR(15)**
  - **@producent NVARCHAR(24)**
  - **@kategoria NVARCHAR(15)**
  - **@ID INT OUTPUT**
- Opis**:**
  - Procedura dodaje nowy produkt do tabeli **Produkty do naprawy** pod warunkiem, że w tabeli **Kategorie** istnieje podana przez argument kategoria.
  - Jeżeli kategoria nie istnieje to zwraca **@ID = NULL**.
  - Jeżeli kategoria istnieje, to wstawia nowy produkt to tabeli **Produkty do naprawy** i zwraca przez **@ID** nowe **ID** tego produktu.

Nowe_zlecenie

- **Argumenty:**
  - **@klient_id INT**
  - **@pracownik_id INT**
  - **@nazwa_produktu NVARCHAR(15)**
  - **@producent NVARCHAR(24)**
  - **@kategoria NVARCHAR(15)**
  - **@koszt MONEY**
  - **@rabat NVARCHAR(50)**
  - **@opis NVARCHAR(255)**
- **Opis:**
  - Procedura tworzy nowe zlecenie w bazie danych na podstawie informacji podanych przez argumenty.
  - Jeżeli procedura nie jest już objęta transakcją, to jest ona uruchamiana w poziomie izolacji **READ COMMITED**.
  - Produkt do naprawy definiowany jest przez **@nazwa_produktu**, **@producent**, **@kategoria** i procedura wyszukuje lub dodaje ten produkt do bazy danych.
  - Jeżeli nazwa rabatu jest poprawna, to rabat jest przypisywany do zlecenia.
  - Procedura wylicza cenę na podstawie **@koszt**, **@rabat** i indywidualnego rabatu klienta.
  - Mając już wszystkie potrzebne informacje, wstawia rekord do tabeli **Zlecenia**, oraz 1 rekord do tabeli **Przebieg zlecen** informujący o rozpoczęciu zlecenia.
- **Może wyrzucić błędy:**
  - **50002**
  - **50003**
  - **50004**
  - **50005**

Kod:

```
CREATE PROCEDURE Nowe_zlecenie(
	@klient_id INT,
	@pracownik_id INT,
	@nazwa_produktu nvarchar(15),
	@producent nvarchar(24),
	@kategoria nvarchar(15),
	@koszt MONEY,
	@rabat nvarchar(50),
	@opis nvarchar(255)
	)
AS
BEGIN TRY
	DECLARE @tranCount INT = @@TRANCOUNT

	IF @tranCount =0
		BEGIN TRAN noweZlecenie

	IF @klient_id NOT IN ( SELECT ID_klienta FROM Klienci)
		RAISERROR(50002, -1, -1)--klient nie istnieje

	IF @pracownik_id NOT IN ( SELECT ID_pracownika FROM Pracownicy)
		RAISERROR(50003, -1, -1)--pracownik nie istnieje

	DECLARE @ID INT--produktu

	SELECT @ID = dbo.Znajdz_produkt(@nazwa_produktu,@producent,@kategoria)

	IF @ID IS NULL
	BEGIN
		EXEC dbo.Nowy_produkt @nazwa_produktu,@producent,@kategoria,@ID OUTPUT

		IF @ID IS NULL
			RAISERROR(50004, -1, -1)--produktu nie bylo w bazie i nie da sie -
			--go dodac bo kategoria jest niepoprawna
	END

	DECLARE @rabat_id INT
	DECLARE @rabat_id_2 INT
	DECLARE @rab REAL

	IF @rabat IS NOT NULL
	BEGIN
		SELECT @rabat_id=R.ID_rabatu, @rab = R.[Procent znizki]
		FROM RABATY AS R
		WHERE R.Nazwa = @rabat

		IF @rabat_id IS NULL
			RAISERROR(50005, -1, -1)--podany rabat jest bledny


		SET @koszt = @koszt * (1- @rab)
	END

	SELECT @rabat_id_2= K.Rabat_Indywidualny
	FROM Klienci AS K
	WHERE K.ID_klienta = @klient_id

	IF @rabat_id_2 IS NOT NULL
	BEGIN
		SELECT @rab=R.[Procent znizki]
		FROM Rabaty AS R
```

```
            WHERE R.ID_rabatu = @rabat_id_2

            SET @koszt = @koszt * (1- @rab)

      END

      INSERT INTO Zlecenia(ID_klienta,ID_pracownika,ID_produktu,Koszt,
                           [Data przyjecia],Opis,ID_rabatu)
      VALUES(@klient_id,@pracownik_id,@ID,@koszt,GETDATE(),@opis,@rabat_id)


      INSERT INTO [Przebieg zlecen](ID_zlecenia,Data,Opis)
      VALUES(@@IDENTITY,GETDATE(),N'Przyjecie zlecenia')

      IF @tranCount = 0
            COMMIT TRAN noweZlecenie

END TRY
BEGIN CATCH
      ROLLBACK TRAN noweZlecenie
      EXEC Bledy
END CATCH
```

Nowe_zlecenie_gwarancyjne

- **Argumenty:**
  - **@gwarancja_id INT**
  - **@klient_id INT**
  - **@pracownik_id INT**
  - **@opis NVARCHAR(255)**
- **Opis:**
  - Procedura tworzy nowe zlecenie z zerowym kosztem, ponieważ jest to zlecenie gwarancyjne.
  - Jeżeli procedura nie jest już objęta transakcją, to jest ona uruchamiana w poziomie izolacji **READ COMMITED**.
  - Każda gwarancja opisuje jakiś produkt, więc produkt nie jest podawany argumentem.
  - Mając już wszystkie potrzebne informacje, wstawia rekord do tabeli **Zlecenia**, oraz 1 rekord do tabeli **Przebieg zlecen** informujący o rozpoczęciu zlecenia.
- **Może wyrzucić błędy:**
  - **50002**
  - **50003**
  - **50010**

Kod:

```
CREATE PROCEDURE Nowe_zlecenie_gwarancyjne(
	@gwarancja_id INT,
	@klient_id INT,
	@pracownik_id INT,
	@opis nvarchar(255)
)
AS
BEGIN TRY
	DECLARE @tranCount INT = @@TRANCOUNT

	IF @tranCount =0
		BEGIN TRAN noweZlecenieGwarancyjne

	IF @klient_id NOT IN ( SELECT ID_klienta FROM Klienci)
		RAISERROR(50002, -1, -1)--klient nie istnieje

	IF @pracownik_id NOT IN ( SELECT ID_pracownika FROM Pracownicy)
		RAISERROR(50003, -1, -1)--pracownik nie istnieje

	IF @gwarancja_id NOT IN ( SELECT ID_gwarancji FROM Gwarancje )
		RAISERROR(50010, -1, -1)--gwarancja nie istnieje

	DECLARE @produkt_id INT

	SELECT @produkt_id = G.ID_produktu
	FROM Gwarancje AS G
```

```
        WHERE G.ID_gwarancji = @gwarancja_id

        INSERT INTO Zlecenia(ID_klienta,ID_pracownika,ID_produktu,Koszt,
                             [Data przyjecia],Opis,ID_rabatu,ID_gwarancji)
      VALUES(@klient_id,@pracownik_id,@produkt_id,0,GETDATE(),@opis,NULL,@gwarancja_id)


        INSERT INTO [Przebieg zlecen](ID_zlecenia,Data,Opis)
        VALUES(@@IDENTITY,GETDATE(),N'Przyjecie zlecenia z gwarancji')

        IF @tranCount = 0
              COMMIT TRAN noweZlecenieGwarancyjne

END TRY
BEGIN CATCH
        ROLLBACK TRAN noweZlecenieGwarancyjne
        EXEC Bledy
END CATCH
```

Nowa_reklamacja

- *Argumenty:*
  - **@zlecenie_id INT**
  - **@klient_id INT**
  - **@pracownik_id INT**
  - **@opis NVARCHAR(255)**
- *Opis:*
  - Procedura tworzy nowe zlecenie z zerowym kosztem, ponieważ jest to reklamacja źle zrealizowanego zlecenia.
  - Jeżeli procedura nie jest już objęta transakcją, to jest ona uruchamiana w poziomie izolacji **READ COMMITED**.
  - Wyszukuje one zlecenie wskazywane przez **@zlecenie_id** i tworzy nowe zlecenie na ten sam produkt, oraz umieszcza informację o reklamacji w tabeli **Reklamacje**.
- *Może wyrzucić błędy:*
  - **50002**
  - **50003**
  - **50006**

Kod:

```
CREATE PROCEDURE Nowa_reklamacja(
	@zlecenie_id INT,
	@klient_id INT,
	@pracownik_id INT,
	@opis nvarchar(255)
)
AS
BEGIN TRY
	DECLARE @tranCount INT = @@TRANCOUNT

	IF @tranCount =0
		BEGIN TRAN nowaReklamacja

	IF @klient_id NOT IN ( SELECT ID_klienta FROM Klienci)
		RAISERROR(50002, -1, -1)--klient nie istnieje

	IF @pracownik_id NOT IN ( SELECT ID_pracownika FROM Pracownicy)
		RAISERROR(50003, -1, -1)--pracownik nie istnieje

	DECLARE @ID INT--produktu

	SELECT @ID = Z.ID_produktu
	FROM Zlecenia AS Z
	WHERE Z.ID_zlecenia=@zlecenie_id

	IF @ID IS NULL
		RAISERROR(50006, -1, -1)--zlecenia nie ma w bazie
```

```
        INSERT INTO Zlecenia(ID_klienta,ID_pracownika,ID_produktu,Koszt,
                            [Data przyjecia],Opis,ID_rabatu)
        VALUES(@klient_id,@pracownik_id,@ID,0,GETDATE(),@opis,NULL)

        SELECT @ID = @@IDENTITY

        INSERT INTO [Przebieg zlecen](ID_zlecenia,Data,Opis)
        VALUES(@ID,GETDATE(),N'Przyjecie reklamacji')

        INSERT INTO Reklamacje(ID_zlecenia,
                            [Nowe ID_zlecenia],ID_pracownika,ID_klienta,Data,Opis)
        VALUES(@zlecenie_id,@ID,@pracownik_id,@klient_id,GETDATE(),@opis)

        IF @tranCount = 0
               COMMIT TRAN nowaReklamacja

END TRY
BEGIN CATCH
        ROLLBACK TRAN nowaReklamacja
        EXEC Bledy
END CATCH
```

Transfer_miedzy_placowkami

- **Argumenty:**
  - **@placowka INT**
  - **@placowka_docelowa INT**
  - **@id_czesci INT**
  - **@ile INT**
- **Opis:**
  - Procedura przesyła części z jednej placówki do drugiej o ile w placówce znajduje się wystarczająco dużo części (**@ile**).
  - Jeżeli procedura nie jest już objęta transakcją, to jest ona uruchamiana w poziomie izolacji **READ COMMITED**.
- **Może wyrzucić błędy:**
  - **50007**
  - **50008**
  - **50009**
  - **50011**
  - **50012**

Kod:

```
CREATE PROCEDURE Transfer_miedzy_placowkami(
	@placowka INT,
	@placowka_docelowa INT,
	@id_czesci INT,@ile INT
)
AS
BEGIN TRY
	DECLARE @tranCount INT = @@TRANCOUNT

	IF @tranCount =0
		BEGIN TRAN transferMiedzyPlacowkami

		IF @placowka NOT IN ( SELECT ID_placowki FROM Placowki)
			RAISERROR(50007, -1, -1)--placowka nie istnieje

		IF @placowka_docelowa NOT IN ( SELECT ID_placowki FROM Placowki)
			RAISERROR(50008, -1, -1)--placowka docelowa nie istnieje

		IF @id_czesci NOT IN (SELECT ID_czesci FROM [Czesci do naprawy])
			RAISERROR(50009, -1, -1)--czesc nie istnieje

		DECLARE @stan INT

		SELECT @stan= SMC.Ilosc
		FROM [Stan magazynowy czesci] AS SMC
		WHERE SMC.ID_placowki=@placowka AND SMC.ID_czesci = @id_czesci

		IF @stan IS NULL
			RAISERROR(50011, -1, -1)--Brak czesci w placowce
		IF @stan < @ile
			RAISERROR(50012, -1, -1)--za malo czesci w placowce
```

```
		UPDATE [Stan magazynowy czesci]
		SET Ilosc= Ilosc - @ile
		WHERE ID_placowki=@placowka AND ID_czesci = @id_czesci

		DELETE FROM [Stan magazynowy czesci] WHERE Ilosc = 0

		SET @stan = NULL

		SELECT @stan= SMC.Ilosc
		FROM [Stan magazynowy czesci] AS SMC
		WHERE SMC.ID_placowki=@placowka_docelowa AND SMC.ID_czesci = @id_czesci

		IF @stan IS NULL--sprawdzam czy placowka docelowa ma dane o tej czesci
		BEGIN
			INSERT INTO [Stan magazynowy czesci]
			VALUES (@placowka_docelowa,@id_czesci,@ile)
			END
		ELSE--jak ma to dodaje do ilosci
		BEGIN
			UPDATE [Stan magazynowy czesci]
			SET Ilosc= Ilosc + @ile
			WHERE ID_placowki=@placowka_docelowa AND ID_czesci = @id_czesci
		END

	IF @tranCount = 0
		COMMIT TRAN transferMiedzyPlacowkami

END TRY
BEGIN CATCH
	ROLLBACK TRAN transferMiedzyPlacowkami
	EXEC Bledy
END CATCH
```

Zamawianie

- **Argumenty:**
  - **@czesc INT**
  - **@placowka INT**
  - **@ile INT**
- **Opis:**
  - Procedura tworzy zamówienie na części podane argumentem.
  - Jeżeli istnieje już zamówienie z tego samego dnia to procedura podpina zamawiane części pod to zamówienie.
  - W przeciwnym wypadku tworzy nowe zamówienie składające się z części podanych argumentem.
  - Cenę i dostawcę części pobiera z tabeli **Czesci**.
  - Jeżeli procedura nie jest już objęta transakcją, to jest ona uruchamiana w poziomie izolacji **READ COMMITED**.
- **Może wyrzucić błędy:**
  - **50007**
  - **50009**

Kod:

```
CREATE PROCEDURE Zamawianie(
	@czesc INT,
	@placowka INT,
	@ile INT
)
AS
BEGIN TRY
	DECLARE @tranCount INT = @@TRANCOUNT

	IF @tranCount =0
		BEGIN TRAN zamawianie

	IF @czesc NOT IN (SELECT ID_czesci FROM [Czesci do naprawy])
			RAISERROR(50009, -1, -1)--czesc nie istnieje

	IF @placowka NOT IN ( SELECT ID_placowki FROM Placowki)
			RAISERROR(50007, -1, -1)--placowka nie istnieje

	DECLARE @hurtownia INT
	DECLARE @cena MONEY
	SELECT @hurtownia = CDN.ID_hurtowni, @cena = CDN.Cena
	FROM [Czesci do naprawy] AS CDN
	WHERE CDN.ID_czesci = @czesc

	DECLARE @zamowienie INT

	SELECT @zamowienie= Z.ID_zamowienia
	FROM Zamowienia AS Z
	WHERE Z.ID_hurtowni = @hurtownia AND Z.Data_zamowienia = CAST(GETDATE() AS DATE)
AND Z.ID_placowki = @placowka
```

```
	IF @zamowienie IS NULL --nie bylo dzisiaj takiego zamowienia
	BEGIN
		INSERT INTO Zamowienia
		(ID_hurtowni,ID_placowki,Data_zamowienia,Data_dostarczenia)
		VALUES(@hurtownia,@placowka,GETDATE(),NULL)

		SET @zamowienie = @@IDENTITY

		INSERT INTO [Szczegoly zamowien]
		VALUES(@zamowienie,@czesc,@cena,@ile)
	END
	ELSE--bylo dzisiaj zamowienie
	BEGIN
		DECLARE @ile_juz_bylo INT -- dla sprawdzenia czy moze juz byla ta czesc
					--dzisiaj zamawiana i tylko zwiekszymy ilosc

		SELECT @ile_juz_bylo = SZ.Ilosc
		FROM [Szczegoly zamowien] AS SZ
		WHERE @zamowienie= SZ.ID_zamowienia AND @czesc=SZ.ID_czesci

		IF @ile_juz_bylo IS NULL --dzisiaj nie bylo tej czesci w zamowieniu
		BEGIN
			INSERT INTO [Szczegoly zamowien]
			VALUES(@zamowienie,@czesc,@cena,@ile)
		END
		ELSE --dopisujemy ilosc
		BEGIN
			UPDATE [Szczegoly zamowien]
			SET Ilosc = Ilosc + @ile
		WHERE @zamowienie= ID_zamowienia AND @czesc=ID_czesci
		END
	END

	IF @tranCount = 0
		COMMIT TRAN zamawianie

END TRY
BEGIN CATCH
	ROLLBACK TRAN zamawianie
	EXEC Bledy
END CATCH
```

Nowa_gwarancja

- **Argumenty:**
  - **@nazwa_produktu NVARCHAR(15)**
  - **@producent NVARCHAR(24)**
  - **@kategoria NVARCHAR(15)**
  - **@klient_id INT**
  - **@pracownik_id INT**
  - **@gwarancja NVARCHAR(50)**
- **Opis:**
  - Procedura tworzy gwarancję dla klienta **@klient_id**, **@pracownik_id** sprzedaje gwarancję.
  - Jeżeli procedura nie jest już objęta transakcją, to jest ona uruchamiana w poziomie izolacji **READ COMMITED**.
  - **@gwarancja** opisuje typ gwarancji.
  - Jeżeli produkt opisany przez **@nazwa_produktu**, **@producent** i **@kategoria** nie istnieje, to zostanie utworzony.
- **Może wyrzucić błędy:**
  - **50002**
  - **50003**
  - **50004**
  - **50010**

Kod:

```
CREATE PROCEDURE Nowa_gwarancja(
	@nazwa_produktu nvarchar(15),
	@producent nvarchar(24),
	@kategoria nvarchar(15),
	@klient_id INT,
	@pracownik_id INT,
	@gwarancja NVARCHAR(50)
)
AS
BEGIN TRY
	DECLARE @tranCount INT = @@TRANCOUNT

	IF @tranCount =0
		BEGIN TRAN nowaGwarancja

	IF @klient_id NOT IN ( SELECT ID_klienta FROM Klienci)
		RAISERROR(50002, -1, -1)--klient nie istnieje

	IF @pracownik_id NOT IN ( SELECT ID_pracownika FROM Pracownicy)
		RAISERROR(50003, -1, -1)--pracownik nie istnieje

	DECLARE @kategoria_id INT

	SELECT @kategoria_id = K.ID_kategorii
```

```
	FROM Kategorie AS K
	WHERE K.Nazwa= @kategoria

	IF @kategoria_id IS NULL
		RAISERROR(50004, -1, -1)--to nie obslugujemy takiej kategorii

	DECLARE @gwarancja_id INT
	DECLARE @dlugosc_gwarancji INT

	SELECT @gwarancja_id = TG.ID_typu_gwarancji,
		@dlugosc_gwarancji=TG.[Czas trwania]
	FROM [Typ gwarancji] AS TG
	WHERE TG.Nazwa = @gwarancja AND TG.ID_kategorii = @kategoria_id

	IF @gwarancja_id IS NULL
		RAISERROR(50010, -1, -1)--to gwarancja nie istnieje

	DECLARE @produkt_id INT

	SELECT @produkt_id = dbo.Znajdz_produkt(@nazwa_produktu,@producent,@kategoria)

	IF @produkt_id IS NULL
		EXEC dbo.Nowy_produkt @nazwa_produktu, @producent, @kategoria,
		@produkt_id OUTPUT

	DECLARE @data DATE
	SET @data = GETDATE()

	INSERT INTO Gwarancje(ID_produktu, ID_klienta, ID_pracownika, [Typ gwarancji],
[Data rozpoczecia], [Data zakonczenia])
	VALUES (@produkt_id, @klient_id, @pracownik_id, @gwarancja_id, @data,
	DATEADD(dd, @dlugosc_gwarancji,@data))

	IF @tranCount = 0
		COMMIT TRAN nowaGwarancja

END TRY
BEGIN CATCH
	ROLLBACK TRAN nowaGwarancja
	EXEC Bledy
END CATCH
```

Bledy

- **Opis:**
  - Procedura wyrzuca (**RAISERROR**) ostatni złapany błąd.

Kod:

```
CREATE PROCEDURE Bledy
AS
	DECLARE @ErrorMessage NVARCHAR(4000);
    DECLARE @ErrorSeverity INT;
    DECLARE @ErrorState INT;

    SELECT
        @ErrorMessage = ERROR_MESSAGE(),
        @ErrorSeverity = ERROR_SEVERITY(),
        @ErrorState = ERROR_STATE()

    RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState);
```

Insert_or_update

- Argumenty:
  - **@Do_dodania Czesci**(typ własny)
- Opis:
  - Procedura dodaje lub aktualizuje (jeśli nie istniały) części w magazynie zdefiniowanym przez Argument.
  - Argument przechowuje **ID_placowki**, **ID_czesci** oraz **Ilosc**.

Kod:

```
CREATE PROCEDURE Insert_or_update(@Do_dodania Czesci READONLY)
AS
BEGIN
	DECLARE @Ilosc_do_dodania INT = (SELECT top 1 Ilosc FROM @Do_dodania)
	DECLARE	@Placowka INT = (SELECT TOP 1 ID_placowki FROM @Do_dodania)
	DECLARE @ID_czesci INT = (SELECT TOP 1 ID_czesci FROM @Do_dodania)

	IF EXISTS (SELECT * FROM [Stan magazynowy czesci] WHERE ID_placowki = @Placowka
AND ID_czesci = @ID_czesci)
		UPDATE [Stan magazynowy czesci] SET
			Ilosc = Ilosc + @Ilosc_do_dodania
		WHERE ID_placowki = @Placowka
		AND ID_czesci = @ID_czesci;
	ELSE
		INSERT INTO [Stan magazynowy czesci] (ID_placowki, ID_czesci, Ilosc)
			VALUES (@Placowka, @ID_czesci, @Ilosc_do_dodania);
END
```

WstawAlboEdytujOsoby

- **Argumenty:**
  - **@ID_osoby INT**
  - **@Imie NVARCHAR(50)**
  - **@Nazwisko NVARCHAR(50)**
  - **@Plec CHAR(1)**
  - **@Adres NVARCHAR(60)**
  - **@NumerKontaktowy VARCHAR(24)**
  - **@Email NVARCHAR(60)**
- **Opis:**
  - Dodaje do bazy danych, lub edytuje Osobę określaną przez argumenty.

Kod:

```
CREATE PROCEDURE WstawAlboEdytujOsoby(
	@ID_osoby INT,
	@Imie NVARCHAR(50),
	@Nazwisko NVARCHAR(50),
	@Plec CHAR(1),
	@Adres NVARCHAR(60),
	@NumerKontaktowy VARCHAR(24),
	@Email NVARCHAR(60))
AS
	IF @ID_osoby = 0
		INSERT INTO Osoby (Imie,Nazwisko,Plec,Adres,[Numer kontaktowy],[E-mail])
		VALUES (@Imie,@Nazwisko,@Plec,@Adres,@NumerKontaktowy,@Email)
	ELSE
		UPDATE Osoby
		SET
			Imie = @Imie,
			Nazwisko = @Nazwisko,
			Plec = @Plec,
			Adres = @Adres,
			[Numer kontaktowy] = @NumerKontaktowy,
			[E-mail] = @Email
		WHERE ID_osoby = @ID_osoby
```

UsunOsoby

- **Argumenty:**
  - **@ID_osoby INT**
- **Opis:**
  - Usuwa z bazy danych osobę o podanym przez argument ID.

Kod:

```
CREATE PROCEDURE UsunOsoby(@ID_osoby INT)
AS
	DELETE FROM Osoby
	WHERE ID_Osoby = @ID_osoby
```

Wyzwalacze

Przenies_czesci

- **Cel i działanie:**
  - Celem tego **triggera** jest przeniesienie części z jednej placówki (usuwanej) do drugiej (nieusuwanej).
  - Przenosi części, tylko jeżeli istnieje placówka która nie jest usuwana, oraz jeżeli nie narusza warunków integralnościowych w innych tabelach.
- **Typ i tabela:**
  - Założony jest na tabelę **Placowki**.
  - Wyzwalacz jest typu **INSTEAD OF DELETE**.
- **Dodatkowe informacje:**
  - Trigger korzysta z typu tabelowego **Czesci**, oraz z procedury **Insert_or_delete**.

Kod:

```
CREATE TRIGGER Przenies_czesci
ON Placowki
INSTEAD OF DELETE
AS
BEGIN
	--Wybieramy placówkę do której wyślemy części
	DECLARE @Placowka_odbiorca INT;
	SET @Placowka_odbiorca = (SELECT TOP 1 ID_placowki
		FROM Placowki
	WHERE ID_placowki NOT IN (SELECT ID_placowki FROM deleted));

	--Jeżeli nie ma placówki-odbiorcy, to znaczy że usuwamy wszystkie placówki.
	IF @Placowka_odbiorca IS NULL
	BEGIN
		DELETE Placowki WHERE ID_placowki IN (SELECT ID_placowki FROM deleted);
		RETURN;
	END

	--Zadeklarujmy potrzebne rzeczy
	DECLARE @Przenoszone_czesci Czesci;
	DECLARE @VAL1 INT;
	DECLARE @VAL2 INT;
	DECLARE Iterator CURSOR
		FOR (--Wybieramy części do przerucenia bez podziału na placówki.
			SELECT S.ID_czesci, SUM(Ilosc)
				FROM deleted AS D
				JOIN [Stan magazynowy czesci] AS S
				ON (S.ID_placowki = D.ID_placowki)
			GROUP BY ID_czesci
			)
		FOR READ ONLY;

	--Iterujmy
	OPEN Iterator;
```

```
	FETCH Iterator INTO @VAL1, @VAL2;
	INSERT INTO @Przenoszone_czesci VALUES (@Placowka_odbiorca, @VAL1,@VAL2);

	WHILE @@FETCH_STATUS = 0
	BEGIN
		BEGIN TRY
			--Przenieśmy części
			EXEC Insert_or_update @Do_dodania = @Przenoszone_czesci;

			--Kolejna iteracja
			DELETE FROM @Przenoszone_czesci;
			FETCH Iterator INTO @VAL1, @VAL2;
				INSERT INTO @Przenoszone_czesci VALUES
		(@Placowka_odbiorca, @VAL1, @VAL2);

		END TRY
		BEGIN CATCH
			ROLLBACK
			EXEC Bledy
			BREAK
		END CATCH
	END

	--Zamykamy iterator
	CLOSE Iterator;
	DEALLOCATE Iterator;

	--Usuwamy części z placówek
	DELETE FROM [Stan magazynowy czesci]
	WHERE ID_placowki IN (SELECT ID_placowki FROM deleted)

	--Usuwamy placówki
	DELETE FROM Placowki
	WHERE ID_placowki IN (SELECT ID_placowki FROM deleted)
END
```

Blokada_aktualizacji_zlecen

- **Cel:**
  - Celem tego **triggera** jest zablokowanie aktualizowania **przebiegu zleceń**, jeżeli zlecenie zostało już zakończone (**Data zrealizowania** jest **Nullem**).
- **Typ:**
  - Założony jest na tabelę **Przebieg zlecen**.
  - Wyzwalacz jest typu **AFTER INSERT**.
- **Dodatkowe informacje:**
  - Może wyrzucić błąd **50001**.

Kod:

```
CREATE TRIGGER Blokada_aktualizacji_zlecen
ON [Przebieg zlecen]
AFTER INSERT
AS
BEGIN
	DECLARE @data DATE;
	--Sprawdzenie czy zlecenie zostało już zakończone
	SET @data = (SELECT TOP 1 Z.[Data zrealizowania]
				FROM Zlecenia AS Z
				JOIN inserted AS I ON (Z.ID_zlecenia =
I.ID_zlecenia)
			WHERE Z.[Data zrealizowania] IS NOT NULL) --Jeśli wszystkie
będą NULLem to zapytanie nic nie zwróci (czyli NULL)

	IF @data IS NOT NULL
	BEGIN
		RAISERROR(50001, -1, -1);
	END
END
```

Stworz_klienta

- **Cel:**
  - Celem tego **triggera** jest automatyczne stworzenie klienta, po dodaniu nowego rekordu do tabeli **osoby**.
- **Typ:**
  - Założony jest na tabelę **Osoby**.
  - Trigger jest typu **AFTER INSERT**.
- **Dodatkowe informacje:**
  - Każdy dodany klient nie posiada rabatu.

Kod:

```
CREATE TRIGGER Stworz_klienta
ON Osoby
AFTER INSERT
AS
BEGIN
	INSERT INTO Klienci(ID_Klienta, Rabat_Indywidualny, Data_Rejestracji)
	SELECT I.ID_osoby, NULL, GETDATE()
		FROM inserted AS I
END
```

Przeslij_czesci_po_zakonczeniu_zamowienia

- **Cel:**
  - Celem tego **triggera** jest przesłanie wszystkich części z właśnie zrealizowanych zamówień (zamiana **Data_dostarczenia** z **NULL'a** na jakąś datę) do placówek na które zostały one zlecone.
  - Jeżeli data była już wpisana i użytkownik będzie chciał ją zamienić na **NULL'a**, to operacja ta zostanie przerwana ponieważ takie działanie mogło by doprowadzić do wielokrotnego przesyłania do magazynu tych samych części.
- **Typ:**
  - Założony jest na tabelę **Zamowienia**.
  - Wyzwalacz jest typu **AFTER UPDATE**.
- **Dodatkowe informacje:**
  - Może wyrzucić błąd **50013**.
  - Trigger korzysta z typu tabelowego **Czesci**, oraz z procedury **Insert_or_delete**.

Kod:

```
CREATE TRIGGER Przeslij_czesci_po_zakonczeniu_zamowienia
ON Zamowienia
AFTER UPDATE
AS
BEGIN
	--Jeżeli użytkownik próbuje zmienić datę na NULL to operacja powinna zostać przerwana.
	--Gdyby użytkownik mógł zmienić na NULL'a to wtedy mógłby znowu ustawić datę zakończenia co ponownie przesłałoby towary.
	IF EXISTS (SELECT *
		--FROM deleted AS D
		--JOIN inserted AS I ON (D.ID_zamowienia = I.ID_zamowienia)
		FROM Zamowienia AS D
		JOIN Zamowienia aS I ON (D.ID_zamowienia = I.ID_zamowienia)
		WHERE D.Data_dostarczenia IS NOT NULL AND I.Data_dostarczenia IS NULL
		)
	BEGIN
		RAISERROR(50013, -1, -1);
		RETURN;
	END

	--Zadeklarujmy pomocnicze zmienne
	DECLARE @Przenoszone_czesci Czesci;
	DECLARE @VAL1 INT;
	DECLARE @VAL2 INT;
	DECLARE @VAL3 INT;
	DECLARE Iterator CURSOR
		FOR (
			SELECT I.ID_placowki, SZ.ID_czesci, SUM(SZ.Ilosc)
				FROM inserted AS I
				JOIN deleted AS D ON (D.ID_zamowienia = I.ID_zamowienia)
				JOIN [Szczegoly zamowien] AS SZ
				ON (SZ.ID_zamowienia = I.ID_zamowienia)
```

```
                WHERE D.Data_dostarczenia IS NULL
                AND I.Data_dostarczenia IS NOT NULL --Sprawdzamy czy zamówienie
                                                    --zostało zrealizowane
                GROUP BY I.ID_placowki, SZ.ID_czesci
        )
        FOR READ ONLY;


    --Iterujemy po wszystkich częściach
    OPEN Iterator;
    FETCH Iterator INTO @VAL1, @VAL2, @VAL3;
    INSERT INTO @Przenoszone_czesci VALUES (@VAL1, @VAL2, @VAL3);

    WHILE @@FETCH_STATUS = 0
    BEGIN
        BEGIN TRY
            --Przenosimy części
            EXEC Insert_or_update @Do_dodania = @Przenoszone_czesci;

            --Wybieramy kolejne czesci
            DELETE FROM @Przenoszone_czesci;
            FETCH Iterator INTO @VAL1, @VAL2, @VAL3;
            INSERT INTO @Przenoszone_czesci VALUES (@VAL1, @VAL2, @VAL3);

        END TRY
        BEGIN CATCH
            ROLLBACK
            EXEC Bledy
            BREAK
        END CATCH
    END

    --Zamykamy iterator
    CLOSE Iterator;
    DEALLOCATE Iterator;
END
```

Blokada_zwolnienia_pracownika_z_niezrealizowanymi_zleceniami

- **Cel:**
  - Celem tego **triggera** jest zablokowanie oznaczenia pracownika jako zwolnionego, jeżeli realizuje on jakieś nieskończone jeszcze zlecenie.
- **Typ:**
  - Założony jest na tabelę **Pracownicy**.
  - Wyzwalacz jest typu **AFTER UPDATE**.
- **Dodatkowe informacje:**
  - Może wyrzucić błąd **50014**.

Kod:

```
CREATE TRIGGER Blokada_zwolnienia_pracownika_z_niezrealizowanymi_zleceniami
ON Pracownicy
AFTER UPDATE
AS
BEGIN
	IF EXISTS (
		SELECT I.ID_pracownika
			FROM inserted AS I
			JOIN deleted AS D ON (I.ID_pracownika = D.ID_pracownika)
			JOIN Zlecenia AS Z ON (Z.ID_pracownika = I.ID_pracownika)
		--Sprawdzamy czy pracownik został zwolniony
		WHERE (D.[Data zwolnienia] IS NULL AND I.[Data zwolnienia] IS NOT NULL)
		AND (Z.[Data zrealizowania] IS NULL)
	)
	BEGIN
		ROLLBACK;
		RAISERROR(50014, -1, -1);
	END
END
```

Typy własne

Czesci

- **Typ tabelowy postaci:**

ID_placowki	ID_czesci	Ilosc
INT NOT NULL	INT NOT NULL	INT NOT NULL CHECK >= 0

- **Zastosowanie:**
  - Argument w procedurze **Insert_or_update**.
  - Stosowany w wyzwalaczach:
    - **Przenies_czesci**
    - **Przeslij_czesci_po_zakonczeniu_zamowienia**

Kod:

```
CREATE TYPE Czesci AS TABLE
(
	ID_placowki INT NOT NULL,
	ID_czesci INT NOT NULL,
	Ilosc INT NOT NULL CHECK(Ilosc >= 0)
);
```

Kody błędów

Kod błędu	Treść
50001	„Zlecenie zostało już zakończone, nie możesz zaktualizować przebiegu”
50002	„Nieprawidlowy klient”
50003	„Nieprawidlowy pracownik”
50004	„Nieprawidlowa kategoria”
50005	„Nieprawidlowy rabat”
50006	„Nieprawidlowe zlecenie”
50007	„Nieprawidlowa placowka”
50008	„Nieprawidlowa placowka docelowa”
50009	„Nieprawidlowa czesc”
50010	„Nieprawidlowa gwarancja”
50011	„Brak czesci w placówce”
50012	„Za malo części w placowce”
50013	„Nie możesz zmienić daty zakończenia zamówienia z powrotem na NULL!”
50014	„Pracownik nie ukonczyl wszystkich zlecen!”

Strategia pielęgnacji bazy danych

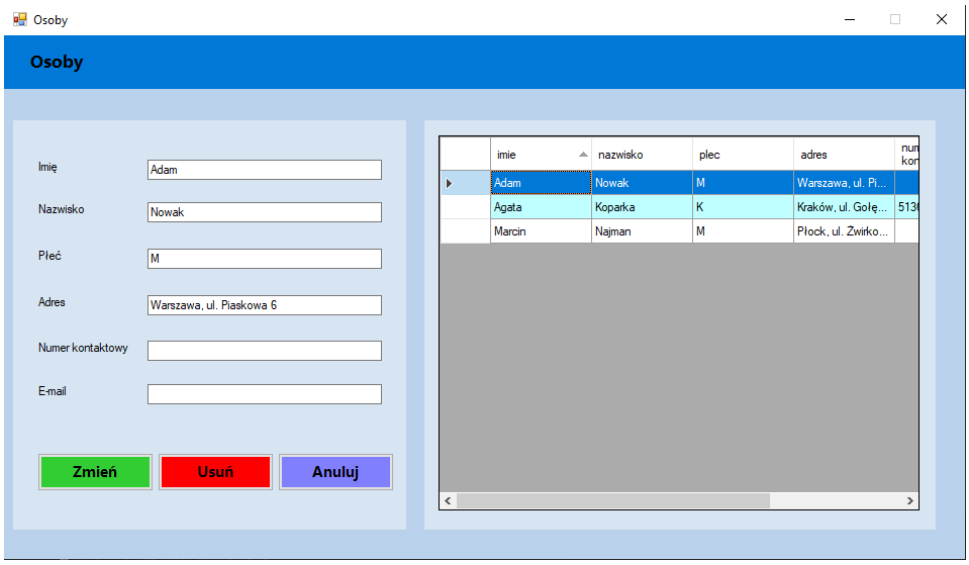
- Pełna kopia zapasowa Bazy Danych będzie tworzona co 4 dni.
- Kopia różnicowa będzie tworzona co 1 dzień.
- Co 30 minut kopia zapasowa dziennika transakcji.

Baza danych, ze względu na swoją specyfikację szybko może urosnąć do dużych rozmiarów, więc pełna kopia zapasowa jest tworzona tylko co 4 dni, a różnicowa co 1.

Ze względu na kluczowość dziennika transakcji, jego kopia jest robiona co 30 minut.

Aplikacja

Dodatkowym elementem projektu jest aplikacja obsługująca podstawowe działania (wyświetlanie, dodawanie, edytowanie, usuwanie) na tabeli **Osoby.**



Aplikacja po prawej stronie wyświetla aktualną zawartość tabeli **Osoby** z pominięciem wartości ID_osoby. Korzystając z textboxów i zielonego przycisku “Zapisz” możemy dodawać kolejne rekordy do tabeli. Edytowanie rekordów odbywa się poprzez podwójne kliknięcie na wybrany rekord, co sprawia, że textboxy wypełniają się wartościami rekordu, a zielony przycisk zamiast “Zapisz” wyświetla “Zmień”. Analogicznie do edytowania możemy usunąć rekord poprzez podwójne jego kliknięcie oraz wybranie opcji “Usuń”. Przycisk “Anuluj” czyści zawartość textboxów. Aplikacja wyświetla również komunikaty dotyczące przebiegu podejmowanych przez użytkownika działań. Komunikat o błędzie pojawi się w przypadku wpisania niepoprawnych danych podczas zapisu lub edycji oraz w przypadku chęci usunięcia osoby, która jest pracownikiem lub jako klient podejmowała jakieś działania jak np. utworzenie zlecenia naprawy.