

Stacja Paliw

Dokumentacja



Krzysztof Kaniuka – 240697

Miłosz Wyrębkiewicz – 240840

Igor Tylak - 240827

Spis treści

[Opis projektu 2](#_Toc1300082650)

[Założenia: 3](#_Toc548979765)

[Ograniczenia 3](#_Toc898852524)

[Baza danych tabele 3](#_Toc1331874965)

[Zależności między tabelami 3](#_Toc1543310300)

[Role 4](#_Toc1901092447)

[Podział źródeł danych 4](#_Toc1896177683)

[Połączenie pomiędzy Microsoft SQL Server oraz Oracle 4](#_Toc686791273)

[Połączenie pomiędzy Microsoft SQL Server oraz Excel 6](#_Toc1729690829)

[Połączenie pomiędzy serwerami Oracle 7](#_Toc607925331)

[Link prywatny: 7](#_Toc1347685446)

[Link publiczny: 8](#_Toc424668174)

[Użytkownicy 8](#_Toc597047566)

[ORACLE 8](#_Toc66924090)

[MS SQL SERVER 11](#_Toc1423485512)

[Utworzone table 16](#_Toc1514840735)

[ SQL Server (instancja stacji) 17](#_Toc1084442691)

[Transakcje paliwowe: 17](#_Toc980881847)

[Transakcje spożywcze: 17](#_Toc373425988)

[Harmonogram: 18](#_Toc393735810)

[ SQL Server (zarząd) 18](#_Toc1717496104)

[ Oracle (zaopatrzenie) 18](#_Toc329306308)

[Dostawcy paliwowi: 19](#_Toc185439651)

[Dostawcy spożywczy: 20](#_Toc126454760)

[Produkty spożywcze: 20](#_Toc1498956000)

[Typ paliwa: 21](#_Toc1449638756)

[Historia cen paliw: 21](#_Toc1607861776)

[Paliwa: 21](#_Toc1830444954)

[Klienci: 22](#_Toc1461988151)

[Zamowienia spożywcze: 22](#_Toc1485652226)

[Zamówienia paliwowe: 23](#_Toc353027422)

[Programy mycia: 23](#_Toc1588372666)

[Transakcje myjni: 24](#_Toc246483532)

[Procedury i funkcje 24](#_Toc538946476)

[Menadżer 24](#_Toc2115501538)

[Procedury 24](#_Toc834630723)

[ Hire\_employee 24](#_Toc752172550)

[ add\_shift\_employee 25](#_Toc884990670)

[ Delete\_employee\_shift 26](#_Toc1989077721)

[ show\_employee\_schedule 26](#_Toc797701289)

[Funkcje 27](#_Toc1682226346)

[ is\_shift\_valid 27](#_Toc753965546)

[ check\_employee\_on\_shift 28](#_Toc1844661459)

[Widoki 29](#_Toc2130776430)

[ Show\_employees 29](#_Toc2058649630)

[Pracownik 29](#_Toc37682820)

[Procedury 29](#_Toc1105359120)

[ update\_product\_stock 29](#_Toc119363921)

[ update\_petrol\_stock 30](#_Toc1814928234)

[ add\_product\_transaction 32](#_Toc663009479)

[ add\_petrol\_transacton 34](#_Toc655447597)

[ make\_product\_order 37](#_Toc210651826)

[ make\_petrol\_order 38](#_Toc993695911)

[ Register\_client 41](#_Toc180343838)

[Funkcje 41](#_Toc986666816)

[Widoki 41](#_Toc356395489)

[ products\_prices 41](#_Toc1330155679)

[ products\_stock 42](#_Toc1080544598)

[ petrol\_prices 42](#_Toc527137362)

[ petrol\_stock 42](#_Toc649792640)

[ show\_order\_fuel\_history 43](#_Toc1116863181)

[Klient 43](#_Toc2090102686)

[Procedury 43](#_Toc2104367806)

[ client\_petrol\_transaction\_history 43](#_Toc911449065)

[ client\_products\_transaction\_history 44](#_Toc458692374)

[ Check\_loyalty\_points 45](#_Toc713353180)

[Funkcje 46](#_Toc1018430338)

[Techniczne 46](#_Toc312084351)

[Funkcje: 46](#_Toc2056545696)

[Procedury 46](#_Toc111197734)

[ Add\_Points\_to\_customer 46](#_Toc1712636761)

[Widoki 47](#_Toc198927466)

[ Rozszerzone\_Transakcje\_Myjni 47](#_Toc715929348)

[Zarząd 47](#_Toc2103822500)

[Funkcje 47](#_Toc1050066516)

[Procedury 47](#_Toc1533130029)

[ set\_product\_price 47](#_Toc258679853)

[ set\_product\_points 48](#_Toc1706483430)

[ show\_transaction\_analize\_daily\_fuel 49](#_Toc2008271739)

[ show\_transaction\_analize\_products\_daily 50](#_Toc205155834)

[ show\_history\_of\_fuel\_price 51](#_Toc1103671891)

[ show\_orders\_of\_fuel 51](#_Toc1490097902)

[ show\_number\_of\_new\_customer 51](#_Toc411036091)

[ update\_petrol\_price 52](#_Toc1723068960)

[ update\_petrol\_points 53](#_Toc1730236102)

[ show\_statistic\_car\_wash 55](#_Toc1495024327)

[ show\_employee\_hours 55](#_Toc1907156468)

[ show\_employee\_works\_all 56](#_Toc2007840928)

[Widoki 56](#_Toc1881159545)

[ view\_price\_history 57](#_Toc963420805)

[Replikacja danych 57](#_Toc282942893)

[Triggery 59](#_Toc1569224416)

[ Rozszerzone\_Transakcje\_Myjni\_trigger 59](#_Toc253683067)

[ on\_petrol\_delivery 60](#_Toc1515681504)

[ on\_products\_delivery 61](#_Toc1382681959)

# Opis projektu

Baza danych dla stacji paliw jest zaprojektowana w celu zarządzania wszystkimi aspektami działalności stacji paliw, w tym sprzedażą paliw oraz produktów, zarządzaniem zapasami produktów spożywczych oraz samych paliw, obsługą klientów, rejestrowaniem transakcji oraz generowaniem raportów.

# Założenia:

1. Maks 3 lub 4 pracowników na zmianie
2. Ten sam pracownik nie może mieć 2 zmian jednego dnia
3. Maksymalnie 1 dostawa danego typu jednego dnia
4. Ilość paliwa nie może spać poniżej 5% (wywala błąd bazy, nie można zrealizować transakcji)
5. Nie możemy przechowywać większej ilości paliwa niż wartość maksymalna do niego przypisana
6. Transakcje mogą być rejestrowane dla klientów bez programu lojalnościowego (KlientID = Null)
7. Mechanizm zliczania liczby punktów na podstawie transakcji (procedura)
8. Zmiana ceny produktu nie może być większa niż 50% poprzedniej cen
9. Redukcja produktów w momencie wykonania zamówienia spożywczego (błąd gdy brakuje produktów w magazynie)

# Ograniczenia

1. Opodatkowanie produktów
2. Regulacje prawne związane z pensją pracowników
3. Zamówienia składane ręcznie (nie zakładamy automatycznego harmonogramu zamówień)
4. Brak rozbudowanej struktury systemu punktów lojalnościowych
5. Brak sklepu punktów lojalnościowych

# Baza danych tabele

* Paliwa
* Pracownicy
* Dostawcy paliwowi
* Dostawcy spożywczy
* Produkty spożywcze
* Zamówienia paliwowe
* Zamówienia spożywcze
* Klienci (karty lojalnościowe)
* Transakcje paliwowe
* Transakcje spożywcze
* Harmonogram zmian

# Zależności między tabelami

* Zamówienia paliwowe – Paliwa (1:\*)
* Zamówienie spożywcze – produkty spożywcze (1:\*)
* Klient (karty lojalnościowe) – transakcje paliwowe (1:\*)
* Klient (karty lojalnościowe) – transakcje spożywcze (1:\*)
* Dostawcy paliwowi – zamówienia paliwowe (1:\*)
* Dostawcy spożywczy – zamówienia spożywcze(1:\*)
* Harmonogram zmian

# Role

* Menadżer stacji (zarządzanie pracownikami i cenami na stacji)
* Pracownik stacji (Sprawdza ilość produktów na magazynie i paliwa, składa zamówienia, realizuje transakcje)
* Klient (sprawdza liczbę punktów)
* Zarząd (prowadzenie analizy, wgląd w statystyki)

# Podział źródeł danych

1. Excel
   1. Pracownicy (mało, łatwo się wykonuje operacje księgowe)
2. Oracle (zaopatrzenie)
   1. Zamówienia spożywcze i paliwowe
   2. Dostawy spożywcze i paliwowe
   3. Klienci (system lojalnościowy) – nieprzypisani do stacji
   4. Produkty spożywcze i paliwa
3. SQL Server (instancja stacji)
   1. Transakcje paliwowe i spożywcze
   2. Harmonogram zmian
4. SQL Server (zarząd) (być  może jako jeden SQL Server)\*
   1. Widoki z transakcji
   2. Średnia okresowa
   3. Maksymalna wartość transakcji
   4. Ilość transakcji jednego dnia, wartość etc.

# Połączenie pomiędzy Microsoft SQL Server oraz Oracle

Utworzone połączenie pomiędzy Microsoft SQL Server oraz Oracle. W ten sposób zostało zaprezentowane, możliwość jaką daje tworzenie linkerów do zasobu innego producenta bazy danych, z którego może korzystać nasz usługodawca stacji paliw.

Aby rozpocząć możliwość przetwarzania danych z Oracle, wymagane jest zainstalowanie sterownika OraOLEDB.Oracle dla serwera MS SQL Server, który pozwoli na przetwarzanie danych znajdujących się na serwerze Oracle.

Kolejnym krokiem jest konfiguracja SQL Server:

sp\_configure 'show\_advanced\_options', 1

GO

RECONFIGURE WITH OverRide

GO

sp\_configure 'Ad Hoc Distributed Queries', 1

GO

RECONFIGURE WITH OverRide

GO

EXEC master.dbo.sp\_MSset\_oledb\_prop N'OraOLEDB.Oracle', N'AllowInProcess', 1

GO

EXEC master.dbo.sp\_MSset\_oledb\_prop N'OraOLEDB.Oracle', N'DynamicParameters', 1

GO

Następnie tworzymy linker do serwera Oracle:

EXEC master.dbo.sp\_addlinkedserver

@server = N'ZaopatrzenieOracle',

@provider = N'OraOLEDB.Oracle', --sterownik OLEDB który znajdziemy w server object

@srvproduct = N'Oracle',

@datasrc = N'(DESCRIPTION =

(ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.0.14)(PORT = 1521))

(CONNECT\_DATA =

(SERVER = DEDICATED)

(SID = orcl)

)

)' --wykorzysstujemy uchwyt TNS (sprawdzamy w NET MANAGER)

Tutaj warto zwrócić uwagę na to, aby serwer Oracle miał poprawnie skonfigurowany Listener, który odpowiada za to, że nasze polecenia docierają do serwera Oracle. Taką konfigurację możemy przeprowadzić w narzędziu udostępnionym przez Oracle: “Net Manager”.

Następnym krokiem jest stworzenie login dla linkera, aby uzyskać dostęp do zasobów Oracle:

EXEC master.dbo.sp\_addlinkedsrvlogin

@rmtsrvname = N'ZaopatrzenieOracle',

@useself = N'False',

@locallogin = N'sa',

@rmtuser = N'AdministratorOracle',

@rmtpassword = N'123'

Ostatecznym krokiem jest skonfigurowanie SQL Servera, aby umożliwić wywoływanie procedur na serwerze Oracle:

EXEC master.dbo.sp\_serveroption @server=N'ZaopatrzenieOracle', @optname=N'rpc', @optvalue=N'true'

GO

EXEC master.dbo.sp\_serveroption @server=N'ZaopatrzenieOracle', @optname=N'rpc out', @optvalue=N'true'

GO

Poprawność konfiguracji sprawdzamy wywołując polecenie:

***SELECT \* FROM ZaopatrzenieOracle.."ADMINISTRATORORACLE"."PALIWA"***

# Połączenie pomiędzy Microsoft SQL Server oraz Excel

Utworzone połączenie pomiędzy Microsoft SQL Server oraz Microsoft Excel, pozwala na dostęp do danych, które mogą być przetwarzane zarówno za pomocą SQL jak i arkusza kalkulacyjnego.

W Excelu w przypadku stacji paliw przechowywane są dane na temat pracowników. Umożliwi to później łatwiejsze przetwarzanie danych np. dla księgowej.

Należy pamiętać o wstępnej konfiguracji po stronie SQL Servera, aby wszystko działało poprawnie. Należy również zwrócić uwagę, aby mieć zainstalowany sterownik OLEDB, który umożliwia połączenie z zasobem:

sp\_configure 'show\_advanced\_options', 1

GO

RECONFIGURE WITH OverRide

sp\_configure 'Ad Hoc Distributed Queries', 1

GO

RECONFIGURE WITH OverRide

GO

EXEC master.dbo.sp\_MSset\_oledb\_prop N'Microsoft.ACE.OLEDB.16.0', N'AllowInProcess', 1

GO

EXEC master.dbo.sp\_MSset\_oledb\_prop N'Microsoft.ACE.OLEDB.16.0', N'DynamicParameters', 1

GO

W pierwszej kolejności tworzymy linker do zasobu:

EXEC master.dbo.sp\_addlinkedserver

@server = N'Pracownicy',

@srvproduct=N'OLE DB Provider for ACE 16.0',

@provider=N'Microsoft.ACE.OLEDB.16.0',

@datasrc=N'C:\Pracownicy.xlsx',

@provstr=N'Excel 12.0;HDR=YES'

GO

Następnie tworzymy login:

EXEC sp\_addlinkedsrvlogin

@rmtsrvname = N'Pracownicy',

@locallogin = N'sa',

@useself = N'False',

@rmtuser = N'admin',

@rmtpassword = N'';

GO

W ten sposób uzyskujemy połączenie pomiędzy MS SQL Server, a arkuszem Excel. Dla sprawdzenia poprawności konfiguracji wywołujemy polecenie: ***SELECT \* FROM Pracownicy...[Pracownicy$];***

# Połączenie pomiędzy serwerami Oracle

Utworzone połączenie pomiędzy serwerami Oracle za pomocą linkowania, pozwala na pozyskiwanie danych z różnych serwerów producenta Oracla. Do połączenia pomiędzy schematami Oracla wykorzystane zostały: link publiczny oraz link prywatny.

## Link prywatny:

CREATE DATABASE LINK StacjaPrywatny

CONNECT TO AdministratorOracle IDENTIFIED BY "123"

USING '(DESCRIPTION =

(ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))

(CONNECT\_DATA =

(SERVER = DEDICATED)

(SID = orcl) ) )';

--polaczenie z linku prywatnego

SELECT \* from klienci@StacjaPrywatny;

Jest on wykorzystywany do między innymi pobierania danych o klientach.

## Link publiczny:

Stworzony link publiczy umożliwia wszystkich użytkownikom pobieranie danych.

CREATE PUBLIC DATABASE LINK StacjaPubliczny

CONNECT TO AdministratorOracle IDENTIFIED BY "123"

USING '(DESCRIPTION =

(ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))

(CONNECT\_DATA =

(SERVER = DEDICATED)

(SID = orcl) ) )';

# Użytkownicy

## ORACLE

Stworzeni użytkownicy pomagają zaimprowizować rzeczywiste role jakie można przyjąć podczas działania stacji paliw.

-- instancja: STACJA PALIW

/\*Administrator\*/

CREATE USER AdministratorOracle IDENTIFIED BY 123;

GRANT ALL PRIVILEGES TO AdministratorOracle;

/\*Menadzer\*/

CREATE USER MenadzerOracle IDENTIFIED BY 123;

GRANT CREATE SESSION TO MenadzerOracle;

GRANT CONNECT TO MenadzerOracle;

GRANT SELECT ON ADMINISTRATORORACLE.DOSTAWCY\_PALIWOWI TO MenadzerOracle;

GRANT SELECT ON ADMINISTRATORORACLE.DOSTAWCY\_SPOZYWCZY TO MenadzerOracle;

GRANT SELECT ON ADMINISTRATORORACLE.HISTORIA\_CEN\_PALIW TO MenadzerOracle;

GRANT SELECT ON ADMINISTRATORORACLE.KLIENCI TO MenadzerOracle;

GRANT SELECT ON ADMINISTRATORORACLE.PALIWA TO MenadzerOracle;

GRANT SELECT ON ADMINISTRATORORACLE.PRODUKTY\_SPOZYWCZE TO MenadzerOracle;

GRANT SELECT ON ADMINISTRATORORACLE.TYP\_PALIWA TO MenadzerOracle;

GRANT SELECT ON ADMINISTRATORORACLE.ZAMOWIENIA\_PALIWOWE TO MenadzerOracle;

GRANT SELECT ON ADMINISTRATORORACLE.ZAMOWIENIA\_SPOZYWCZE TO MenadzerOracle;

GRANT EXECUTE ON ADMINISTRATORORACLE.set\_product\_points TO MenadzerOracle;

GRANT EXECUTE ON ADMINISTRATORORACLE.set\_product\_price TO MenadzerOracle;

GRANT EXECUTE ON ADMINISTRATORORACLE.update\_petrol\_points TO MenadzerOracle;

GRANT EXECUTE ON ADMINISTRATORORACLE.update\_petrol\_price TO MenadzerOracle;

/\*Pracownik\*/

CREATE USER PracownikOracle IDENTIFIED BY 123;

GRANT CREATE SESSION TO PracownikOracle;

GRANT CONNECT TO PracownikOracle;

GRANT SELECT ON ADMINISTRATORORACLE.DOSTAWCY\_PALIWOWI TO PracownikOracle;

GRANT SELECT ON ADMINISTRATORORACLE.DOSTAWCY\_SPOZYWCZY TO PracownikOracle;

GRANT SELECT ON ADMINISTRATORORACLE.HISTORIA\_CEN\_PALIW TO PracownikOracle;

GRANT SELECT ON ADMINISTRATORORACLE.KLIENCI TO PracownikOracle;

GRANT SELECT ON ADMINISTRATORORACLE.PALIWA TO PracownikOracle;

GRANT SELECT ON ADMINISTRATORORACLE.PRODUKTY\_SPOZYWCZE TO PracownikOracle;

GRANT SELECT ON ADMINISTRATORORACLE.TYP\_PALIWA TO PracownikOracle;

GRANT SELECT ON ADMINISTRATORORACLE.ZAMOWIENIA\_PALIWOWE TO PracownikOracle;

GRANT SELECT ON ADMINISTRATORORACLE.ZAMOWIENIA\_SPOZYWCZE TO PracownikOracle;

GRANT UPDATE ON ADMINISTRATORORACLE.PRODUKTY\_SPOZYWCZE TO PracownikOracle;

GRANT UPDATE ON ADMINISTRATORORACLE.KLIENCI TO PracownikOracle;

GRANT UPDATE ON ADMINISTRATORORACLE.PALIWA TO PracownikOracle;

GRANT INSERT ON ADMINISTRATORORACLE.ZAMOWIENIA\_PALIWOWE TO PracownikOracle;

GRANT INSERT ON ADMINISTRATORORACLE.ZAMOWIENIA\_SPOZYWCZE TO PracownikOracle;

GRANT INSERT ON ADMINISTRATORORACLE.KLIENCI TO PracownikOracle;

GRANT EXECUTE ON ADMINISTRATORORACLE.UPDATE\_PETROL\_STOCK TO PracownikOracle;

GRANT EXECUTE ON ADMINISTRATORORACLE.UPDATE\_PRODUCT\_STOCK TO PracownikOracle;

/\*Klient\*/

CREATE USER KlientOracle IDENTIFIED BY 123;

GRANT CREATE SESSION TO KlientOracle;

GRANT CONNECT TO KlientOracle;

GRANT SELECT ON ADMINISTRATORORACLE.KLIENCI TO KlientOracle;

/\*Zarzad\*/

CREATE USER ZarzadOracle IDENTIFIED BY 123;

GRANT CREATE SESSION TO ZarzadOracle;

GRANT CONNECT TO ZarzadOracle;

GRANT SELECT ON ADMINISTRATORORACLE.HISTORIA\_CEN\_PALIW TO ZarzadOracle;

GRANT SELECT ON ADMINISTRATORORACLE.KLIENCI TO ZarzadOracle;

GRANT SELECT ON ADMINISTRATORORACLE.PALIWA TO ZarzadOracle;

GRANT SELECT ON ADMINISTRATORORACLE.PRODUKTY\_SPOZYWCZE TO ZarzadOracle;

GRANT SELECT ON ADMINISTRATORORACLE.TYP\_PALIWA TO ZarzadOracle;

GRANT SELECT ON MYJNIA.ROZSZERZONE\_TRANSAKCJE\_MYJNI TO ZarzadOracle;

GRANT EXECUTE ON ADMINISTRATORORACLE.update\_petrol\_points TO ZarzadOracle;

GRANT EXECUTE ON ADMINISTRATORORACLE.update\_petrol\_price TO ZarzadOracle;

/\*Zarzad\*/

-- instancja: MYJNIA

CREATE USER Myjnia IDENTIFIED BY 123;

GRANT CREATE SESSION TO Myjnia;

GRANT CREATE TABLE TO Myjnia;

GRANT CREATE PROCEDURE TO Myjnia;

GRANT CREATE SEQUENCE TO Myjnia;

GRANT CREATE VIEW TO Myjnia;

GRANT CREATE TRIGGER TO Myjnia;

GRANT CREATE SYNONYM TO Myjnia;

--GRANT ALL PRIVILEGES TO Myjnia;

ALTER USER Myjnia QUOTA unlimited ON users;

--jeżeli wywali blad że uprawienia nie zostaly przyznane to dobrze

REVOKE SELECT ANY TABLE FROM Myjnia;

REVOKE INSERT ANY TABLE FROM Myjnia;

REVOKE UPDATE ANY TABLE FROM Myjnia;

REVOKE DELETE ANY TABLE FROM Myjnia;

GRANT CREATE DATABASE LINK TO Myjnia;

## MS SQL SERVER

/\* Administrator \*/

USE [master]

GO

CREATE LOGIN [Administrator] WITH PASSWORD=N'123', DEFAULT\_DATABASE=[master], CHECK\_EXPIRATION=ON, CHECK\_POLICY=ON

GO

USE [StacjaPaliw]

GO

CREATE USER [Administrator] FOR LOGIN [Administrator] WITH DEFAULT\_SCHEMA=[dbo]

GO

ALTER SERVER ROLE [bulkadmin] ADD MEMBER [Administrator]

GO

USE [StacjaPaliw]

GO

ALTER ROLE [db\_owner] ADD MEMBER [Administrator]

GO

--uprawnienia do procedur i funkcji

use [StacjaPaliw]

GO

GRANT EXECUTE TO Administrator;

USE [master]

GO

CREATE LOGIN [Menadzer] WITH PASSWORD=N'123', DEFAULT\_DATABASE=[master], CHECK\_EXPIRATION=ON, CHECK\_POLICY=ON

GO

USE [StacjaPaliw]

GO

CREATE USER [Menadzer] FOR LOGIN [Menadzer] WITH DEFAULT\_SCHEMA=[dbo]

GO

GRANT SELECT ON dbo.show\_employees TO Menadzer;

GO

GRANT SELECT ON dbo.petrol\_prices TO Menadzer;

GO

GRANT SELECT ON [dbo].[petrol\_stock] TO Menadzer;

GO

GRANT SELECT ON [dbo].[products\_prices] TO Menadzer;

GO

GRANT SELECT ON [dbo].[products\_stock] TO Menadzer;

GO

GRANT EXECUTE ON [dbo].[hire\_employee] TO Menadzer;

GRANT EXECUTE ON [dbo].[add\_shift\_employee] TO Menadzer;

GRANT EXECUTE ON [dbo].[delete\_employee\_shift] TO Menadzer;

GRANT EXECUTE ON [dbo].[show\_employee\_schedule] TO Menadzer;

GRANT EXECUTE ON [dbo].[is\_shift\_valid] TO Menadzer;

GRANT EXECUTE ON [dbo].[check\_employee\_on\_shift] TO Menadzer;

GRANT SELECT ON [dbo].[Harmonogram] TO Menadzer;

/\* Pracownik \*/

USE [master]

GO

CREATE LOGIN [Pracownik] WITH PASSWORD=N'123', DEFAULT\_DATABASE=[StacjaPaliw], CHECK\_EXPIRATION=ON, CHECK\_POLICY=ON

GO

USE [StacjaPaliw]

GO

CREATE USER [Pracownik] FOR LOGIN [Pracownik] WITH DEFAULT\_SCHEMA=[dbo]

GO

GRANT EXECUTE ON [dbo].[add\_product\_transaction] TO Pracownik;

GRANT EXECUTE ON [dbo].[add\_petrol\_transaction] TO Pracownik;

GRANT EXECUTE ON [dbo].[make\_product\_order] TO Pracownik;

GRANT EXECUTE ON [dbo].[make\_petrol\_order] TO Pracownik;

GRANT EXECUTE ON [dbo].[show\_employee\_schedule] TO Pracownik;

GRANT EXECUTE ON [dbo].[add\_points\_to\_client] TO Pracownik;

GRANT EXECUTE ON [dbo].[register\_client] TO Pracownik;

GRANT SELECT ON Harmonogram TO Pracownik;

GRANT SELECT ON Transakcje\_spozywcze TO Pracownik;

GRANT SELECT ON Transakcje\_paliwowe TO Pracownik;

GRANT SELECT ON dbo.petrol\_prices TO Pracownik;

GRANT SELECT ON dbo.petrol\_stock TO Pracownik;

GRANT SELECT ON dbo.products\_prices TO Pracownik;

GRANT SELECT ON dbo.products\_stock TO Pracownik;

--GRANT EXECUTE ON [dbo].[register\_client] TO Pracownik

/\* Klient \*/

USE [master]

GO

CREATE LOGIN [Klient] WITH PASSWORD=N'123', DEFAULT\_DATABASE=[StacjaPaliw], CHECK\_EXPIRATION=ON, CHECK\_POLICY=ON

USE [StacjaPaliw]

GO

CREATE USER [Klient] FOR LOGIN [Klient] WITH DEFAULT\_SCHEMA=[dbo]

GO

GRANT EXECUTE ON [dbo].[check\_loyalty\_points] TO Klient;

GRANT EXECUTE ON [dbo].[client\_petrol\_transaction\_history] TO Klient

GRANT EXECUTE ON [dbo].[client\_products\_transaction\_history] TO Klient

/\*Zarzad\*/

USE [master]

GO

CREATE LOGIN [Zarzad] WITH PASSWORD=N'123', DEFAULT\_DATABASE=[StacjaPaliw], CHECK\_EXPIRATION=ON, CHECK\_POLICY=ON

USE [StacjaPaliw]

GO

CREATE USER [Zarzad] FOR LOGIN [Zarzad] WITH DEFAULT\_SCHEMA=[dbo]

GO

GRANT SELECT ON HARMONOGRAM TO ZARZAD;

GRANT SELECT ON TRANSAKCJE\_PALIWOWE TO ZARZAD;

GRANT SELECT ON TRANSAKCJE\_SPOZYWCZE TO ZARZAD;

GRANT SELECT ON dbo.show\_employees TO ZARZAD;

GO

GRANT SELECT ON dbo.petrol\_prices TO ZARZAD;

GO

GRANT SELECT ON [dbo].[petrol\_stock] TO ZARZAD;

GO

GRANT SELECT ON [dbo].[products\_prices] TO ZARZAD;

GO

GRANT SELECT ON [dbo].[products\_stock] TO ZARZAD;

GO

GRANT SELECT ON [dbo].[view\_price\_history] TO ZARZAD;

GRANT EXECUTE ON [dbo].[show\_transaction\_analize\_daily\_fuel] TO ZARZAD;

GRANT EXECUTE ON [dbo].[show\_transaction\_analize\_products\_daily] TO ZARZAD;

GRANT EXECUTE ON [dbo].[show\_number\_of\_new\_customer] TO ZARZAD;

GRANT EXECUTE ON [dbo].[update\_price\_of\_fuel] TO ZARZAD;

GRANT EXECUTE ON [dbo].[update\_points\_of\_fuel] TO ZARZAD;

GRANT EXECUTE ON [dbo].[show\_statistic\_car\_wash] TO ZARZAD;

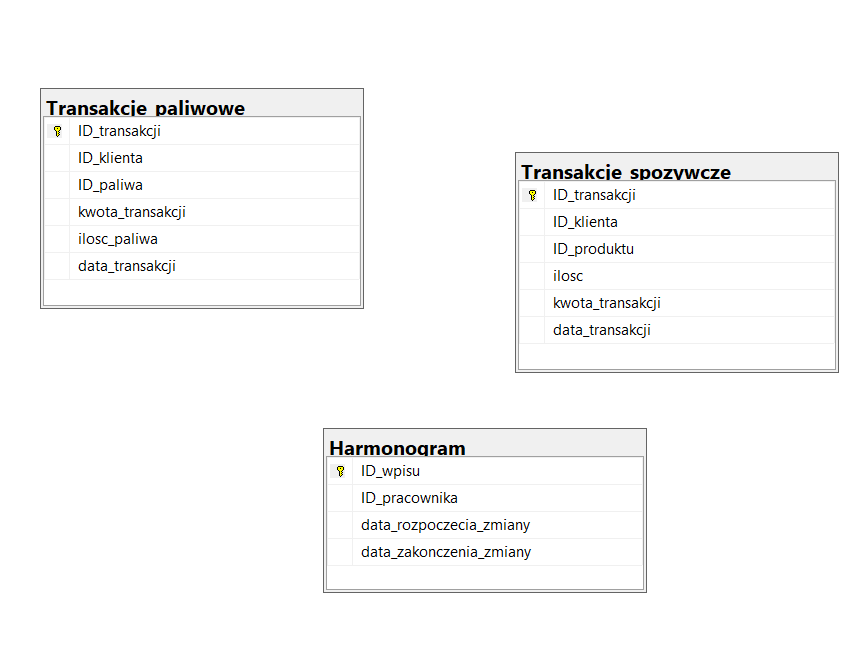
GRANT EXECUTE ON [dbo].[show\_employee\_hours] TO ZARZAD;

GRANT EXECUTE ON [dbo].[show\_employee\_works\_all] TO ZARZAD;

GRANT EXECUTE ON [dbo].[show\_history\_of\_fuel\_price] TO ZARZAD;

# Utworzone table

## SQL Server (instancja stacji)



### Transakcje paliwowe:

Tabela utworzona do przechowywania informacji o transakcjach przeprowadzanych w wyniku sprzedaży paliw.

CREATE TABLE Transakcje\_paliwowe(

ID\_transakcji INT IDENTITY(1,1), --klucz główny

ID\_klienta INT, --klucz obcy klienta

ID\_paliwa INT, --klucz obcy paliwa

kwota\_transakcji NUMERIC(8,2), --kwota przeprowadzonej transakcji

ilosc\_paliwa NUMERIC(5,2), --ilość sprzedaży paliwa

data\_transakcji DATETIME, --data przeprowadzonej transakcji

CONSTRAINT PK\_Transakcje\_paliwowe PRIMARY KEY (ID\_transakcji)

-- ustawienie klucza głównego na pole ID\_transakcji

);

### Transakcje spożywcze:

Tabela stworzona do przechowywania informacji o transakcjach przeprowadzonych w wyniku sprzedaży produktów spożywczych.

CREATE TABLE Transakcje\_spozywcze(

ID\_transakcji INT IDENTITY(1,1), -- klucz główny

ID\_klienta INT, -- klucz obcy klienta

ID\_produktu INT, -- klucz obcy produktu

ilosc INT, -- ilość sprzedawanego produktu

kwota\_transakcji NUMERIC(8,2), -- kwota transakcji

data\_transakcji DATETIME, -- data przeprowadzanej transakcji

CONSTRAINT PK\_Transakcje\_Spozywcze PRIMARY KEY (ID\_transakcji),

-- ustawianie klucza głównego na polu ID\_transakcji

);

### Harmonogram:

Tabela stworzona do przechowywania informacji o zmianach pracowników stacji paliw.

CREATE TABLE Harmonogram

(

ID\_wpisu INT IDENTITY(1,1), -- klucz główny

ID\_pracownika INT, -- klucz obcy pracownika

data\_rozpoczecia\_zmiany DATETIME, -- data rozpoczęcia zmiany pracownika

data\_zakonczenia\_zmiany DATETIME, -- data zakończenia zmiany pracownika

CONSTRAINT PK\_Harmonogram PRIMARY KEY (ID\_wpisu), -- ustawienie klucza głównego na polu ID\_wpisu

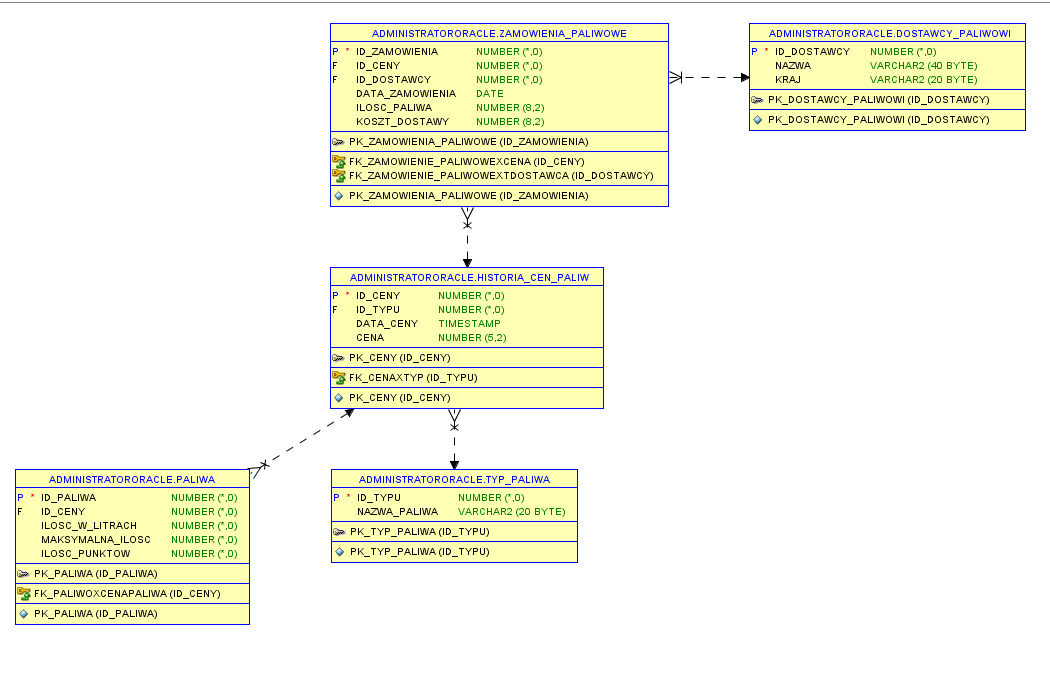
CONSTRAINT CHCK\_ID\_pracownika CHECK (ID\_pracownika IS NOT NULL) -- ustawienie constraint, aby uniknąć null dla pola ID\_pracownika (zmiana bez pracownika).

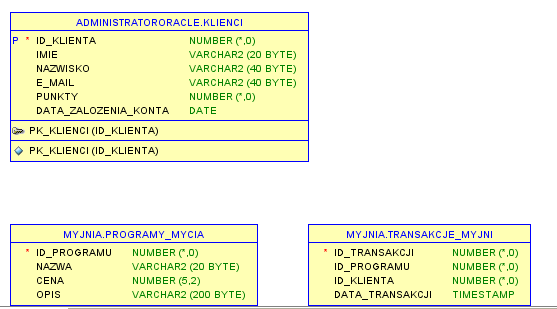
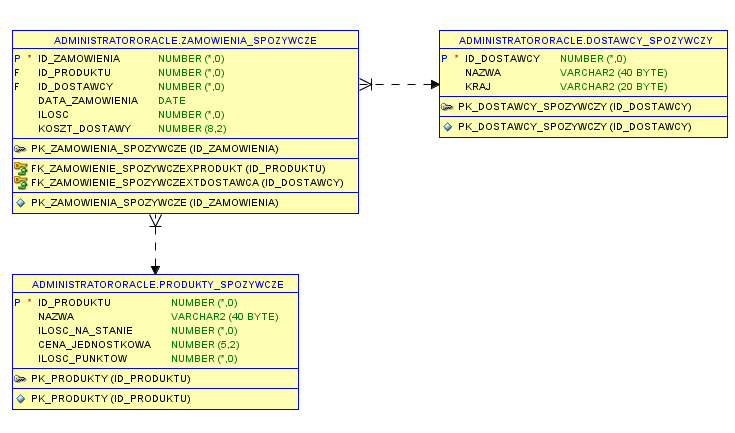
);

## SQL Server (zarząd)

Tabele w bazie danych zarządu, są takie same jak w instancji paliw. Wykorzystano mechanizm replikacji danych pomiędzy instancją stacji paliw, a zarządem.

## Oracle (zaopatrzenie)





### Dostawcy paliwowi:

Tabela stworzona do przechowywania informacji o dostawcach paliwowych.

CREATE TABLE Dostawcy\_Paliwowi(

ID\_dostawcy INTEGER GENERATED ALWAYS AS IDENTITY(START WITH 1 INCREMENT BY 1),

-- klucz główny

nazwa VARCHAR(40), -- pełna nazwa dostawcy/firmy

kraj VARCHAR(20), -- kraj pochodzenia dostawcy

CONSTRAINT PK\_Dostawcy\_Paliwowi PRIMARY KEY(ID\_dostawcy) -- ustawienie klucza głównego na polu ID\_dostawcy

);

### Dostawcy spożywczy:

Tabela stworzona do przechowywania informacji o dostawcach produktów spożywczych.

CREATE TABLE Dostawcy\_Spozywczy(

ID\_dostawcy INTEGER GENERATED ALWAYS AS IDENTITY(START WITH 1 INCREMENT BY 1),

-- klucz główny

nazwa VARCHAR(40), -- pełna nazwa dostawcy

kraj VARCHAR(20), -- kraj pochodzenia dostawcy

CONSTRAINT PK\_Dostawcy\_Spozywczy PRIMARY KEY(ID\_dostawcy) -- ustawienie klucza głównego na polu ID\_dostawcy

);

### Produkty spożywcze:

Tabela stworzona do przechowywania informacji na temat produktów spożywczych sprzedawanych na stacji paliw.

CREATE TABLE Produkty\_spozywcze(

ID\_produktu INTEGER GENERATED ALWAYS AS IDENTITY(START WITH 1 INCREMENT BY 1),

-- klucz główny

nazwa VARCHAR(40), -- pełna nazwa produktu

ilosc\_na\_stanie INTEGER, -- aktualna ilość produktu na stanie magazynownym

cena\_jednostkowa NUMBER(5,2), -- cena sprzedaży produktu

ilosc\_punktow INTEGER, -- ilość punktów, otrzymanych za zakpu produktu

CONSTRAINT PK\_Produkty PRIMARY KEY(ID\_produktu) -- ustawienie klucza głównego na polu ID\_produktu

);

### Typ paliwa:

Tabela stworzona do przechowywania informacji o rodzajach paliw obsługiwanych przez stację paliwową.

CREATE TABLE Typ\_Paliwa(

ID\_typu INTEGER GENERATED ALWAYS as IDENTITY(START with 1 INCREMENT by 1),

-- klucz główny

nazwa\_paliwa VARCHAR(20), -- pełna nazwa rodzaju paliwa

CONSTRAINT PK\_typ\_paliwa PRIMARY KEY (ID\_typu) -- ustawienia klucza głównego na polu ID\_typu

);

### Historia cen paliw:

Tabela stworzona do przechowywania informacji o przeszłych cenach paliw.

CREATE TABLE Historia\_cen\_paliw(

ID\_ceny INTEGER GENERATED ALWAYS as IDENTITY(START with 1 INCREMENT by 1), -- klucz główny

ID\_typu INTEGER, -- klucz obcy do rodzaju paliwa

data\_ceny TIMESTAMP, -- data obejmująca cenę paliwa z danego dnia

cena NUMBER(5,2), -- cena paliwa obowiązująca w danym dniu

CONSTRAINT PK\_Ceny PRIMARY KEY (ID\_ceny), -- ustawienia klucza głównego dla pola ID\_ceny

CONSTRAINT FK\_CenaXTyp FOREIGN KEY (ID\_typu) REFERENCES Typ\_Paliwa(ID\_typu) ON DELETE CASCADE – ustawienie klucza obcego dla pola ID\_typu do tabli Typ\_Paliwa

);

### Paliwa:

Tabela stworzona do przechowywania informacji o paliwie tj. Ilości na stanie, maksymalnej ilość oraz punków zdobytych za zakup paliwa.

CREATE TABLE Paliwa(

ID\_paliwa INTEGER GENERATED ALWAYS as IDENTITY(START with 1 INCREMENT by 1),

-- klucz główny

ID\_ceny INTEGER, -- klucz obcy do tabeli Historia\_cen\_paliwa

ilosc\_w\_litrach INTEGER, -- ilość aktualnego paliwa na stacji

maksymalna\_ilosc INTEGER, -- maksymalna ilość paliwa jaką może przechowywać stacja

ilosc\_punktow INTEGER, --ilość punktów za litr

CONSTRAINT PK\_Paliwa PRIMARY KEY (ID\_paliwa), -- ustawienie klucza głównego na polu ID\_paliwa

CONSTRAINT FK\_PaliwoXCenaPaliwa FOREIGN KEY (ID\_ceny) REFERENCES Historia\_cen\_paliw(ID\_ceny) ON DELETE CASCADE – ustawienie klucza obcego na polu ID\_ceny do tabeli Historia\_cen\_paliw

);

### Klienci:

Tabela stworzona do przechowywania informacji o klientach stacji.

CREATE TABLE Klienci(

ID\_klienta INTEGER GENERATED ALWAYS AS IDENTITY(START WITH 1 INCREMENT BY 1),

-- klucz główny

imie VARCHAR(20), -- imię klienta

nazwisko VARCHAR(40), -- nazwisko klienta

e\_mail VARCHAR(40), -- email klienta

punkty INTEGER, -- aktualna ilość punktów lojalnościowych, które uzbierał klient

data\_zalozenia\_konta DATE, -- data założenia konta lojalnościowego klienta

CONSTRAINT PK\_Klienci PRIMARY KEY(ID\_klienta) -- ustawienie klucza głównego na polu ID\_klienta

);

### Zamowienia spożywcze:

Tabela utworzona do przechowywania informacji o składanych zamówieniach spożywczych.

CREATE TABLE Zamowienia\_Spozywcze(

ID\_zamowienia INTEGER GENERATED ALWAYS AS IDENTITY(START WITH 1 INCREMENT BY 1),

-- klucz główny

ID\_produktu INTEGER, -- klucz obcy do tabeli Produkty\_spożywcze

ID\_dostawcy INTEGER, -- klucz obcy do tabeli Dostawcy\_spożywczy

data\_zamowienia DATE, -- data składania zamówienia

ilosc INTEGER, -- ilość zamawianego produktu

koszt\_dostawy NUMBER(8,2), -- koszt dostawy produktu

CONSTRAINT PK\_Zamowienia\_spozywcze PRIMARY KEY(ID\_zamowienia), -- ustawienie klucza głównego na polu ID\_zamowienia

CONSTRAINT FK\_Zamowienie\_spozywczeXProdukt FOREIGN KEY (ID\_produktu) REFERENCES Produkty\_spozywcze(ID\_produktu), -- ustawienie klucza obcego dla pola ID\_produku do tabeli Produkty\_spożywcze

CONSTRAINT FK\_Zamowienie\_spozywczeXTDostawca FOREIGN KEY (ID\_dostawcy) REFERENCES Dostawcy\_Spozywczy(ID\_dostawcy) -- ustawienie klucza obcego dla pola ID\_dostawcy do tabeli Dostawy\_spozywczy

);

### Zamówienia paliwowe:

Tabela stworzona do przechowywania informacji o składanych zamówieniach paliwowych.

CREATE TABLE Zamowienia\_Paliwowe(

ID\_zamowienia INTEGER GENERATED ALWAYS AS IDENTITY(START WITH 1 INCREMENT BY 1),

-- klucz główny

ID\_ceny INTEGER, -- klucz obcy do tabeli Historia\_cen\_paliw

ID\_dostawcy INTEGER, -- klucz obcy do tabeli Dostawcy\_paliwowi

data\_zamowienia DATE, -- data składanego zamówienia

ilosc\_paliwa NUMBER(8,2), -- ilość zamawianego paliwa

koszt\_dostawy NUMBER(8,2), -- koszt składanego zamówienia

CONSTRAINT PK\_Zamowienia\_paliwowe PRIMARY KEY(ID\_zamowienia), -- ustawienie klucza głównego dla pola ID\_zamowienia

CONSTRAINT FK\_Zamowienie\_paliwoweXCena FOREIGN KEY (ID\_ceny) REFERENCES Historia\_Cen\_Paliw(ID\_ceny) ON DELETE CASCADE, -- ustawienie klucza obcego dla pola ID\_ceny odnoszącego do tabeli Historia\_Cen\_Paliw

CONSTRAINT FK\_Zamowienie\_paliwoweXTDostawca FOREIGN KEY (ID\_dostawcy) REFERENCES Dostawcy\_Paliwowi(ID\_dostawcy) -- ustawienie klucza obcego dla pola ID\_dostawcy odnoszącego do tabeli Dostawcy\_Paliwowi

);

### Programy mycia:

Tabela stworzona do przechowywania informacji o programach mycia pojazdów oferowanych przez stację.

CREATE TABLE Programy\_mycia(

ID\_programu INT GENERATED ALWAYS AS IDENTITY(START WITH 1 INCREMENT BY 1), -- klucz główny

nazwa VARCHAR(20), -- pełna nazwa oferowanego programu

cena NUMBER(5,2), -- cena programu

opis VARCHAR(200) -- dokładny opis programu

);

### Transakcje myjni:

Tabela stworzona do przechowywania informacji o przeprowadzanych transakcjach myjni pojazdów.

CREATE TABLE Transakcje\_myjni(

ID\_transakcji INT GENERATED ALWAYS AS IDENTITY(START WITH 1 INCREMENT BY 1), -- klucz główny

ID\_programu INT, -- klucz obcy do tabeli Programy\_mycia

ID\_klienta INT, -- klucz obcy do tabeli Klienci

data\_transakcji TIMESTAMP – data przeprowadzonej transakcji

CONSTRAINT FK\_Transakcja\_myjniXProgram\_mycia FOREIGN KEY (ID\_programu) REFERENCES Program\_mycia(ID\_programu) ON DELETE CASCADE, -- ustawienie klucza obcego dla pola ID\_programu odnoszącego do tabeli Programy\_mycia.

CONSTRAINT FK\_Transakcja\_myjniXKlienci FOREIGN KEY (ID\_klienta) REFERENCES Klienci(ID\_klienta) -- ustawienie klucza obcego dla pola ID\_klienta odnoszącego do tabeli Klienci

);

# Procedury i funkcje

## Menadżer

### Procedury

#### Hire\_employee

Procedura odpowiedzialna za zatrudnianie nowego pracownika na stacji paliw.

CREATE PROCEDURE hire\_employee(

@emp\_ID INT, -- numer id pracownika

@name VARCHAR(30), -- imie pracownika

@surname VARCHAR(30),-- nazwisko pracownika

@phone\_number VARCHAR(30), -- numer telefonu pracownika

@salary NUMERIC) -- stawka godzinowa pracownika

AS

DECLARE @id\_busy INT

BEGIN

IF @salary < 0

BEGIN

RAISERROR('Wartość pensji nie może być mniejsza od 0', 16,1);

RETURN;

END

SELECT @id\_busy = COUNT(ID\_pracownika) FROM Pracownicy...[Pracownicy$]

WHERE ID\_pracownika = @emp\_ID;

IF @id\_busy > 0

BEGIN

RAISERROR('Istnieje już pracownik o wskazanym ID', 16,1);

RETURN;

END

INSERT INTO OPENQUERY(Pracownicy,'Select \* FROM [Pracownicy$]')

VALUES(@emp\_ID,@name,@surname,@phone\_number,@salary);

END;

#### add\_shift\_employee

Procedura odpowiedzialna za ustawienie zmiany danemu pracownikowi.

CREATE OR ALTER PROCEDURE add\_shift\_employee(

@id\_emp INT, -- numer id pracownika

@shift\_date DATETIME) -- data wraz z czasem startu zmiany pracownika

AS

BEGIN

IF dbo.check\_employee\_on\_shift(@shift\_date) > 4

BEGIN

RAISERROR('Too much employees on shift', 16, 1);

RETURN;

END;

IF dbo.is\_shift\_valid(@id\_emp, @shift\_date) = 0

BEGIN

RAISERROR('Invalid shift time or no employee', 16, 1);

RETURN;

END;

ELSE

BEGIN

INSERT INTO [StacjaPaliw].[dbo].[Harmonogram] ([ID\_pracownika], [data\_rozpoczecia\_zmiany], [data\_zakonczenia\_zmiany])

VALUES (@id\_emp, @shift\_date, DATEADD(HOUR, 8,@shift\_date));

PRINT N'Shift added ' + CAST(@id\_emp AS NVARCHAR(10)) + N' at ' + CAST(@shift\_date AS NVARCHAR(20));

END;

END;

#### Delete\_employee\_shift

Procedura służy do usuwania danej zmiany przypisanej do pracownika.

CREATE OR ALTER PROCEDURE delete\_employee\_shift(

@shift\_id INT) -- numer id wpisu zmiany

AS

BEGIN

DELETE FROM Harmonogram WHERE ID\_wpisu = @shift\_id;

PRINT N'Shift ID: ' + CAST(@shift\_id AS VARCHAR(10)) + N' deleted.'

END;

#### show\_employee\_schedule

Procedura wyświetlająca informacje o zmianach danego pracownika.

CREATE OR ALTER PROCEDURE show\_employee\_schedule(

@emp\_id INT) -- numer id pracownika

AS

BEGIN

SELECT ID\_wpisu, data\_rozpoczecia\_zmiany,data\_zakonczenia\_zmiany FROM Harmonogram

WHERE ID\_pracownika = @emp\_id

ORDER BY data\_rozpoczecia\_zmiany DESC;

END;

### Funkcje

#### is\_shift\_valid

Funkcja sprawdza możliwość ustawienia zmiany dla danego pracownika. Funkcja zwraca wartość -1 w przypadku braku pracownika, 0 gdy zmiany nie można ustawić oraz 1 gdy zmiana może zostać przypisana dla danego pracownika.

CREATE OR ALTER FUNCTION is\_shift\_valid(

@id\_emp INT = 0, -- numer id pracownika

@shift\_date DATETIME) -- data z czasem zmiany pracownika

RETURNS INT

AS

BEGIN

DECLARE @isValid INT;

--search emp

SELECT @isValid = COUNT(ID\_pracownika) FROM Pracownicy...[Pracownicy$] WHERE ID\_pracownika = @id\_emp;

-- if emp not exist return 0

IF @isValid = 0

BEGIN

RETURN 0;

END;

-- implement cursor to check shifts

DECLARE @shift\_start DATETIME;

DECLARE @shift\_end DATETIME;

DECLARE shift\_cursor CURSOR FOR

SELECT data\_rozpoczecia\_zmiany, data\_zakonczenia\_zmiany

FROM dbo.Harmonogram WHERE ID\_pracownika = @id\_emp;

OPEN shift\_cursor;

FETCH NEXT FROM shift\_cursor INTO @shift\_start, @shift\_end;

WHILE @@FETCH\_STATUS = 0

BEGIN

IF @shift\_date BETWEEN DATEADD(HOUR,-8,@shift\_start) AND DATEADD(HOUR,8, @shift\_end)

BEGIN

SET @isValid = 0;

BREAK;

END;

FETCH NEXT FROM shift\_cursor INTO @shift\_start, @shift\_end;

END;

CLOSE shift\_cursor;

DEALLOCATE shift\_cursor;

RETURN @isValid;

END;

#### check\_employee\_on\_shift

Funkcja zwracająca liczbę pracowników znajdujących się na zmianie w danym dniu.

CREATE OR ALTER FUNCTION check\_employee\_on\_shift(

@shift\_time DATETIME) -- data zmiany

RETURNS INT

AS

BEGIN

DECLARE @emp\_number INT;

SELECT @emp\_number = COUNT(ID\_pracownika) FROM Harmonogram WHERE CONVERT(date,data\_rozpoczecia\_zmiany) = CONVERT(DATE,@shift\_time);

RETURN @emp\_number;

END;

### Widoki

* Show\_employees

Widok wyświetlający informację o pracownikach pobranych z zasobów Excel.

CREATE VIEW show\_employees AS

SELECT \* FROM OPENQUERY(Pracownicy,'Select \* from [Pracownicy$]');

GO

select \* from show\_employees;

## Pracownik

### Procedury

#### update\_product\_stock

Procedura wykorzystywana do aktualizacji ilości produktów spożywczych na stanie magazynowym stacji paliw.

CREATE OR REPLACE PROCEDURE update\_product\_stock

(product\_name IN VARCHAR2, -- nazwa produktu spożywczego

new\_stock IN INTEGER) -- nowa ilość produktu na stanie

IS

product\_exist INT;

BEGIN

IF new\_stock < 0 THEN

raise\_application\_error(-20001,'Ilosc produktow na stanie nie moze byc wartoscia ujemna');

RETURN;

END IF;

SELECT COUNT(\*) INTO product\_exist FROM produkty\_spozywcze

WHERE NAZWA = product\_name;

IF product\_exist = 0 THEN

raise\_application\_error(-20001,'Nieprawidlowa nazwa produktu');

RETURN;

END IF;

UPDATE Produkty\_spozywcze SET ILOSC\_NA\_STANIE = new\_stock

WHERE NAZWA = product\_name;

END;

#### update\_petrol\_stock

Procedura wykorzystywana do aktualizacji cen na dany rodzaj paliwa obsługiwanego przez stację paliw.

CREATE OR REPLACE Procedure update\_petrol\_stock

(petrol\_name IN varchar2, -- nazwa rodzaju paliwa

new\_stock IN NUMBER) -- nowa ilość paliwa na stanie

IS

petrol\_name\_exist INTEGER;

petrol\_id INTEGER;

new\_points\_id INTEGER;

restock\_validate INTEGER;

BEGIN

IF new\_stock < 0 THEN

raise\_application\_error(-20001,'Ilosc paliwa na stanie nie moze byc wartoscia ujemna');

RETURN;

END IF;

SELECT COUNT(p.id\_paliwa) INTO petrol\_name\_exist FROM paliwa p

JOIN historia\_cen\_paliw hcp using(id\_ceny)

JOIN typ\_paliwa tp using(id\_typu)

WHERE tp.nazwa\_paliwa = petrol\_name;

IF petrol\_name\_exist = 0 THEN

raise\_application\_error(-20001,'Nieprawidlowa nazwa paliwa');

RETURN;

END IF;

SELECT p.id\_paliwa INTO petrol\_id FROM paliwa p

JOIN historia\_cen\_paliw hcp using(id\_ceny)

JOIN typ\_paliwa tp using(id\_typu)

WHERE tp.nazwa\_paliwa = petrol\_name;

SELECT (MAKSYMALNA\_ILOSC - new\_stock) INTO restock\_validate FROM ADMINISTRATORORACLE.PALIWA

WHERE ID\_PALIWA = petrol\_id;

IF restock\_validate < 0 THEN

raise\_application\_error(-20001,'Nowy stan paliwa nie moze przekraczac wartosci maksymalnej');

RETURN;

END IF;

UPDATE Paliwa SET ilosc\_w\_litrach = new\_stock

WHERE ID\_paliwa = petrol\_id;

COMMIT;

dbms\_output.put\_line('Stan danego paliwa zostal zaktualizowany');

END;

* add\_product\_transaction

Procedura odpowiedzialna za przeprowadzenie transakcji sprzedaży produktu spożywczego.

Jednocześnie usuwa podaną w zamówieniu ilość produktu oraz sprawdza przed tą operacją czy mamy wystarczającą ilość produktów.

CREATE PROCEDURE add\_product\_transaction(

@client\_ID INT, -- numer id klienta

@product\_ID INT, -- numer id produktu

@amount INT ) -- ilość sprzedawanego produktu

AS

DECLARE @client\_exist INT

DECLARE @product\_exist INT

DECLARE @products\_amount INT

DECLARE @products\_value NUMERIC(5,2)

DECLARE @points\_to\_add INT

BEGIN

IF @amount <= 0

BEGIN

RAISERROR('Ilosc produktow musi byc wartoscia dodatnia',16,1);

RETURN;

END

IF @client\_ID IS NOT NULL

BEGIN

SELECT @client\_exist = COUNT(ID\_klienta) FROM OPENQUERY(ZaopatrzenieOracle,'Select \* from ADMINISTRATORORACLE.Klienci')

WHERE ID\_klienta = @client\_ID;

IF @client\_exist = 0

BEGIN

RAISERROR('Klient o podanym ID nie istnieje',16,1);

RETURN;

END

END

SELECT @product\_exist = COUNT(ID\_produktu) FROM OPENQUERY(ZaopatrzenieOracle,'Select \* from ADMINISTRATORORACLE.Produkty\_spozywcze')

WHERE ID\_produktu = @product\_ID;

IF @product\_exist = 0

BEGIN

RAISERROR('Produkt o podanym ID nie istnieje',16,1);

RETURN;

END

SELECT @products\_amount = ILOSC\_NA\_STANIE FROM OPENQUERY(ZaopatrzenieOracle,'Select \* from ADMINISTRATORORACLE.Produkty\_spozywcze')

WHERE ID\_produktu = @product\_ID;

--BRAKUJE PRODUKTÓW DO SKOMPLETOWANIA ZAMÓWIENIA

IF @products\_amount - @amount < 0

BEGIN

RAISERROR('Brakuje produktow do skompletowania zamowienia. Zaleca się zamowienia produktow',16,1);

RETURN;

END

--odjęcie produktów powiązanych z transakcją

UPDATE ZaopatrzenieOracle.."ADMINISTRATORORACLE"."PRODUKTY\_SPOZYWCZE"

SET ILOSC\_NA\_STANIE = ILOSC\_NA\_STANIE - @amount

WHERE ID\_PRODUKTU = @product\_ID

OPTION (RECOMPILE);

SELECT @products\_value = @amount \* CENA\_JEDNOSTKOWA, @points\_to\_add = @amount \* ILOSC\_PUNKTOW

FROM OPENQUERY(ZaopatrzenieOracle,'SELECT \* FROM ADMINISTRATORORACLE.Produkty\_spozywcze')

WHERE ID\_PRODUKTU = @product\_ID;

INSERT INTO Transakcje\_spozywcze(ID\_klienta,ID\_produktu,ilosc,kwota\_transakcji,data\_transakcji)

VALUES(@client\_ID,@product\_ID,@amount,@products\_value,GETDATE());

IF @client\_ID IS NOT NULL

BEGIN

EXEC [dbo].[add\_points\_to\_client]

@client\_ID = @client\_ID,

@points\_to\_add = @points\_to\_add;

END

END;

* add\_petrol\_transacton

Procedura odpowiedzialna za przeprowadzenie transakcji sprzedaży paliwa.

CREATE PROCEDURE add\_petrol\_transaction(

@client\_ID INT, -- numer id klienta

@petrol\_ID INT, -- numer id paliwa

@amount INT ) – ilość paliwa objętego w transakcji

AS

DECLARE @client\_exist INT

DECLARE @petrol\_exist INT

DECLARE @petrol\_amount INT

DECLARE @petrol\_max\_amount INT

DECLARE @petrol\_value NUMERIC(8,2)

DECLARE @points\_to\_add INT

DECLARE @update\_sql NVARCHAR(MAX)

BEGIN

IF @amount <= 0

BEGIN

RAISERROR('Ilosc paliwa musi byc wartoscia dodatnia',16,1);

RETURN;

END

IF @client\_ID IS NOT NULL

BEGIN

SELECT @client\_exist = COUNT(ID\_klienta) FROM OPENQUERY(ZaopatrzenieOracle,'Select \* from ADMINISTRATORORACLE.Klienci')

WHERE ID\_klienta = @client\_ID;

IF @client\_exist = 0

BEGIN

RAISERROR('Klient o podanym ID nie istnieje',16,1);

RETURN;

END

END

SELECT @petrol\_exist = COUNT(ID\_paliwa) FROM OPENQUERY(ZaopatrzenieOracle,'Select \* from ADMINISTRATORORACLE.Paliwa')

WHERE ID\_paliwa = @petrol\_ID;

IF @petrol\_exist = 0

BEGIN

RAISERROR('Paliwo o podanym ID nie istnieje',16,1);

RETURN;

END

SELECT @petrol\_amount = ILOSC\_W\_LITRACH, @petrol\_max\_amount = MAKSYMALNA\_ILOSC FROM OPENQUERY(ZaopatrzenieOracle,'Select \* from ADMINISTRATORORACLE.Paliwa')

WHERE ID\_paliwa = @petrol\_ID;

--KRYTYCZNY POZIOM PALIWA W DYSTRYBUTORZE

IF @petrol\_amount - @amount <= @petrol\_max\_amount \* 0.05 --wartosc ustawiona na sztywno 5% z maksa

BEGIN

RAISERROR('Krytyczny poziom paliwa w dystrybutorze. Zaleca się zamowienie paliwa',16,1);

RETURN;

END

--odjęcie paliwa powiązanego z transakcją

UPDATE ZaopatrzenieOracle..ADMINISTRATORORACLE.PALIWA

SET ILOSC\_W\_LITRACH = ILOSC\_W\_LITRACH - @amount

WHERE ID\_PALIWA = @petrol\_ID;

SELECT @petrol\_value = @amount \* HCP.CENA, @points\_to\_add = @amount \* P.ILOSC\_PUNKTOW

FROM OPENQUERY(ZaopatrzenieOracle,'SELECT \* FROM ADMINISTRATORORACLE.PALIWA') AS P

JOIN OPENQUERY(ZaopatrzenieOracle,'SELECT \* FROM ADMINISTRATORORACLE.HISTORIA\_CEN\_PALIW') AS HCP

ON P.ID\_CENY = HCP.ID\_CENY

WHERE P.ID\_PALIWA = @petrol\_ID;

INSERT INTO Transakcje\_paliwowe(ID\_klienta,ID\_paliwa,kwota\_transakcji,ilosc\_paliwa,data\_transakcji)

VALUES(@client\_ID,@petrol\_ID,@petrol\_value,@amount,GETDATE())

IF @client\_ID IS NOT NULL

BEGIN

EXEC [dbo].[add\_points\_to\_client]

@client\_ID = @client\_ID,

@points\_to\_add = @points\_to\_add;

END

END;

* make\_product\_order

Procedura odpowiedzlana za składanie zamówienia na dany produkt spożywczy, przez danego dostawcę.

CREATE PROCEDURE make\_product\_order(

@product\_name VARCHAR(40), -- nazwa produktu spożywczego

@provider\_ID INT, -- numer id dostawcy

@amount INT, -- ilość zamawianego produktu

@delivery\_cost NUMERIC ) -- koszt dostawy

AS

DECLARE @product\_exist INT

DECLARE @provider\_exist INT

DECLARE @product\_ID INT

BEGIN

IF @amount < 0

BEGIN

RAISERROR('Ilosc paliwa nie moze byc wartoscia ujemna',16,1);

RETURN;

END

IF @delivery\_cost < 0

BEGIN

RAISERROR('Cena dostawy nie moze byc wartoscia ujemna',16,1);

RETURN;

END

SELECT @product\_exist = COUNT(ID\_produktu) FROM ZaopatrzenieOracle.."ADMINISTRATORORACLE"."PRODUKTY\_SPOZYWCZE"

WHERE nazwa = @product\_name;

IF @product\_exist = 0

BEGIN

RAISERROR('Dany produkt nie istnieje',16,1);

RETURN;

END

SELECT @provider\_exist = COUNT(ID\_dostawcy) FROM ZaopatrzenieOracle.."ADMINISTRATORORACLE"."DOSTAWCY\_SPOZYWCZY"

WHERE ID\_DOSTAWCY = @provider\_ID;

SELECT @product\_ID = ID\_produktu

FROM ZaopatrzenieOracle.."ADMINISTRATORORACLE"."PRODUKTY\_SPOZYWCZE"

WHERE nazwa = @product\_name;

IF @provider\_exist = 0

BEGIN

RAISERROR('Dostawca o podanym ID nie istnieje',16,1);

RETURN;

END

INSERT INTO OPENQUERY(ZaopatrzenieOracle,'Select ID\_produktu,ID\_dostawcy,data\_zamowienia,ilosc,koszt\_dostawy FROM ADMINISTRATORORACLE.Zamowienia\_spozywcze')

VALUES(@product\_ID,@provider\_ID,GETDATE(),@amount,@delivery\_cost)

OPTION (RECOMPILE);

END;

#### make\_petrol\_order

Zamówienie na paliwo, sprawdza czy żądana ilość nie przekroczy maksimum i czy nie ma dostawy danego dnia

CREATE PROCEDURE make\_petrol\_order(

@petrol\_name VARCHAR(30), -- nazwa zamawianego paliwa

@provider\_ID INT, -- numer id dostawcy

@petrol\_amout NUMERIC, -- ilość zamawianego paliwa

@delivery\_cost NUMERIC ) -- koszt dostawy

AS

DECLARE @petrol\_exist INT

DECLARE @price\_ID INT

DECLARE @provider\_exist INT

DECLARE @current\_petrol\_amount INT

DECLARE @max\_petrol\_amount INT

BEGIN

IF @petrol\_amout < 0

BEGIN

RAISERROR('Ilosc paliwa nie moze byc wartoscia ujemna',16,1);

RETURN;

END

IF @delivery\_cost < 0

BEGIN

RAISERROR('Cena dostawy nie moze byc wartoscia ujemna',16,1);

RETURN;END

SELECT @petrol\_exist = COUNT(ID\_typu) FROM ZaopatrzenieOracle.."ADMINISTRATORORACLE"."TYP\_PALIWA"

WHERE nazwa\_paliwa = @petrol\_name

IF @petrol\_exist = 0

BEGIN

RAISERROR('Paliwo o wskazanej nazwie nie istnieje', 16,1);

RETURN;

END

SELECT @provider\_exist = COUNT(ID\_dostawcy) FROM ZaopatrzenieOracle.."ADMINISTRATORORACLE"."DOSTAWCY\_PALIWOWI"

WHERE ID\_dostawcy = @provider\_ID

IF @provider\_exist = 0

BEGIN

RAISERROR('Dostawca o wskazanym identyfikatorze nie istnieje', 16,1);

RETURN;

END

SELECT @current\_petrol\_amount = p.ilosc\_w\_litrach,@max\_petrol\_amount = p.maksymalna\_ilosc,@price\_ID = p.ID\_ceny

FROM

OPENQUERY(ZaopatrzenieOracle,'SELECT \* FROM ADMINISTRATORORACLE.PALIWA') AS p

LEFT JOIN OPENQUERY(ZaopatrzenieOracle, 'Select \* from ADMINISTRATORORACLE.HISTORIA\_CEN\_PALIW') as hcp ON p.ID\_ceny=hcp.ID\_ceny

LEFT JOIN OPENQUERY(ZaopatrzenieOracle, 'SELECT \* FROM ADMINISTRATORORACLE.TYP\_PALIWA') as tp ON hcp.ID\_typu = tp.ID\_typu

WHERE tp.nazwa\_paliwa = @petrol\_name

OPTION (RECOMPILE)

IF @current\_petrol\_amount + @petrol\_amout >= @max\_petrol\_amount

BEGIN

RAISERROR('Proba zamówienia zbyt duzej ilosci paliwa', 16,1);

RETURN; END

INSERT INTO OPENQUERY(ZaopatrzenieOracle,'Select ID\_ceny,ID\_dostawcy,data\_zamowienia,ilosc\_paliwa,koszt\_dostawy FROM ADMINISTRATORORACLE.ZAMOWIENIA\_PALIWOWE')

VALUES

(@price\_ID,@provider\_ID,GETDATE(),@petrol\_amout,@delivery\_cost)

OPTION (RECOMPILE);

END;

* Register\_client

Stwórz nowego klienta (wartość punktów 0) na podstawie imienia, nazwiska i maila.

CREATE PROCEDURE register\_client

(@name VARCHAR(40), -- nazwa klienta

@surname VARCHAR(50), -- nazwisko klienta

@email VARCHAR(50) ) -- email klienta

AS

DECLARE @sql NVARCHAR(MAX);

BEGIN

SET @sql = N'INSERT INTO ADMINISTRATORORACLE.KLIENCI (IMIE,NAZWISKO,E\_MAIL,PUNKTY,DATA\_ZALOZENIA\_KONTA)

VALUES ('''+ @name + ''', ''' + @surname + ''', ''' + @email + ''',0,SYSDATE)';

EXEC (@sql) AT ZaopatrzenieOracle;

END;

### Funkcje

### Widoki

#### products\_prices

Widok pobierający informację z serwera Oracle o nazwie, cena obowiązujących produkty spożywcze oraz ilości punktów uzyskanych po zakupie.

CREATE VIEW products\_prices AS

SELECT \* FROM

OPENQUERY(ZaopatrzenieOracle,'SELECT ID\_produktu,nazwa,cena\_jednostkowa,ilosc\_punktow FROM ADMINISTRATORORACLE.produkty\_spozywcze');

GO

SELECT \* FROM products\_prices;

#### products\_stock

Widok pobierający informację z serwera Oracle o produktach spożywczych: nazwę oraz ilość na stanie.

CREATE VIEW products\_stock AS

SELECT \* FROM

OPENQUERY(ZaopatrzenieOracle,'SELECT ID\_produktu,nazwa,ilosc\_na\_stanie FROM ADMINISTRATORORACLE.produkty\_spozywcze');

SELECT \* FROM products\_stock;

* petrol\_prices

Widok wyświetalący informację o cenach oraz ilość punktów obowiązujących na dane paliwo na stacji.

CREATE VIEW petrol\_prices AS

Select p.ID\_paliwa,tp.nazwa\_paliwa,hcp.cena, p.ilosc\_punktow,hcp.data\_ceny

FROM

ZaopatrzenieOracle.."ADMINISTRATORORACLE"."PALIWA" p

LEFT JOIN OPENQUERY(ZaopatrzenieOracle,'SELECT \* FROM ADMINISTRATORORACLE.Historia\_cen\_paliw') hcp

ON p.ID\_ceny = hcp.ID\_ceny

LEFT JOIN ZaopatrzenieOracle.."ADMINISTRATORORACLE"."TYP\_PALIWA" tp

ON tp.ID\_typu = hcp.ID\_typu;

SELECT \* FROM petrol\_prices;

#### petrol\_stock

Widok wyświetlający informację o paliwie min. nazwa oraz o aktualnym stanie magazynowym paliwa na stacji.

CREATE VIEW petrol\_stock AS

Select p.ID\_paliwa,tp.nazwa\_paliwa,p.ilosc\_w\_litrach,p.maksymalna\_ilosc, ROUND((p.ilosc\_w\_litrach/p.maksymalna\_ilosc \* 100),2) as procent

FROM

ZaopatrzenieOracle.."ADMINISTRATORORACLE"."PALIWA" p

LEFT JOIN OPENQUERY(ZaopatrzenieOracle,'SELECT \* FROM ADMINISTRATORORACLE.Historia\_cen\_paliw') hcp

ON p.ID\_ceny = hcp.ID\_ceny

LEFT JOIN ZaopatrzenieOracle.."ADMINISTRATORORACLE"."TYP\_PALIWA" tp

ON tp.ID\_typu = hcp.ID\_typu;

SELECT \* FROM petrol\_stock;

#### show\_order\_fuel\_history

Widok wyświetlający informację o dostawcy, dacie zamówienia i obowiązującej cenie w danym dniu.

CREATE OR REPLACE VIEW show\_order\_fuel\_history

AS

SELECT Zam.id\_zamowienia, Hist.Id\_typu, Hist.cena, Dos\_pal.nazwa ,Zam.data\_zamowienia,

Zam.ilosc\_paliwa, Zam.koszt\_dostawy

FROM zamowienia\_paliwowe Zam

JOIN historia\_cen\_paliw Hist

ON Zam.id\_ceny = Hist.id\_ceny

JOIN dostawcy\_paliwowi Dos\_pal

ON Dos\_pal.id\_dostawcy = Zam.ID\_Dostawcy;

SELECT \* FROM show\_order\_fuel\_history

## Klient

### Procedury

#### client\_petrol\_transaction\_history

Historia transakcji paliwowych klienta.

CREATE PROCEDURE client\_petrol\_transaction\_history

( @client\_ID INT ) -- numer id klienta

AS

BEGIN

DECLARE @client\_petrol\_transactions TABLE

(

ID\_paliwa INT,

kwota\_transakcji SMALLMONEY,

ilosc\_paliwa NUMERIC(5,2),

data\_transakcji DATETIME

);

INSERT INTO @client\_petrol\_transactions(ID\_paliwa,kwota\_transakcji,ilosc\_paliwa,data\_transakcji)

SELECT ID\_paliwa,kwota\_transakcji,ilosc\_paliwa,data\_transakcji

FROM Transakcje\_paliwowe

WHERE ID\_klienta = @client\_ID;

SELECT \* FROM @client\_petrol\_transactions;

END;

#### client\_products\_transaction\_history

Historia transakcji zakupu produktów spożywczych przez klienta.

CREATE PROCEDURE client\_products\_transaction\_history

( @client\_ID INT ) -- numer id klienta

AS

BEGIN

DECLARE @client\_products\_transactions TABLE

(

ID\_produktu INT,

ilosc INT,

kwota\_transakcji SMALLMONEY,

data\_transakcji DATETIME

);

INSERT INTO @client\_products\_transactions(ID\_produktu,ilosc,kwota\_transakcji,data\_transakcji)

SELECT ID\_produktu,ilosc,kwota\_transakcji,data\_transakcji

FROM Transakcje\_spozywcze

WHERE ID\_klienta = @client\_ID;

SELECT \* FROM @client\_products\_transactions;

END;

#### Check\_loyalty\_points

Sprawdzenie ilości zgromadzonych punktów przez danego klienta

CREATE PROCEDURE check\_loyalty\_points(

@client\_ID INT) -- numer id klienta

AS

Declare @client\_exist INT;

Declare @points INT;

BEGIN

SELECT @client\_exist = COUNT(ID\_klienta) FROM ZaopatrzenieOracle.."ADMINISTRATORORACLE"."KLIENCI"

WHERE ID\_klienta = @client\_ID;

IF @client\_exist = 0

BEGIN

RAISERROR('Klient o danym ID nie istnieje', 16,1);

RETURN;

END

SELECT @points = punkty FROM ZaopatrzenieOracle.."ADMINISTRATORORACLE"."KLIENCI"

WHERE ID\_klienta = @client\_ID;

PRINT N'Klient o ID ' + CAST(@client\_ID AS NVARCHAR(10)) + N' posiada ' + CAST(@points AS NVARCHAR(10)) + N' punktów';

END;

### Funkcje

## Techniczne

### Funkcje:

### Procedury

#### Add\_Points\_to\_customer

Procedura odpowiedzialna za dodawanie odpowiedniej ilości punktów na konto danego klienta.

CREATE PROCEDURE add\_points\_to\_client(

@client\_ID INT, -- numer id klienta

@points\_to\_add INT – liczba punktów dodana do konta lojalnościowego

)

AS

DECLARE @client\_exist INT;

BEGIN

SELECT @client\_exist = COUNT(ID\_klienta) FROM ZaopatrzenieOracle.."ADMINISTRATORORACLE"."KLIENCI"

WHERE ID\_klienta = @client\_ID;

IF @client\_exist = 0

BEGIN

RAISERROR('Klient o danym ID nie istnieje', 16,1);

RETURN;

END

UPDATE ZaopatrzenieOracle.."ADMINISTRATORORACLE"."KLIENCI" SET punkty=punkty+@points\_to\_add

WHERE ID\_klienta = @client\_ID

OPTION (RECOMPILE);

END;

### Widoki

#### Rozszerzone\_Transakcje\_Myjni

Widok wyświetlający informację o przeprowadzonych transakcjach na myjni pojazdów wraz z informacjami o kliencie. Widok wykorzystuje linkowanie prywatne utworzone w Oracle.

CREATE OR REPLACE VIEW Rozszerzone\_Transakcje\_Myjni AS

SELECT pm.nazwa nazwa\_programu,k.imie,k.nazwisko,tm.data\_transakcji

FROM Transakcje\_Myjni tm

LEFT JOIN Programy\_Mycia pm USING(ID\_programu)

LEFT JOIN klienci@StacjaPrywatny k USING(ID\_klienta);

## Zarząd

### Funkcje

### Procedury

#### set\_product\_price

Procedura wykorzystywana do aktualizowania cen za danych rodzaj produktu spożywczego.

create or replace PROCEDURE set\_product\_price

(product\_name IN VARCHAR2, -- nazwa produktu

new\_product\_price IN NUMBER) -- nowa cena produktu

IS

product\_name\_exist INT;

previous\_price NUMBER;

BEGIN

IF new\_product\_price < 0 THEN

raise\_application\_error(-20001,'Cena nie moze byc mniejsza od 0');

RETURN;

END IF;

SELECT COUNT(\*) INTO product\_name\_exist FROM produkty\_spozywcze

WHERE NAZWA = product\_name;

IF product\_name\_exist = 0 THEN

raise\_application\_error(-20001,'Nieprawidlowa nazwa produktu');

RETURN;

END IF;

--Sprawdzenie czy zmiana ceny mniejsza niz 50% poprzedniej

SELECT CENA\_JEDNOSTKOWA INTO previous\_price

FROM PRODUKTY\_SPOZYWCZE WHERE

NAZWA = product\_name

FETCH FIRST 1 ROW ONLY;

IF ABS(new\_product\_price - previous\_price) > 0.5 \* previous\_price THEN

raise\_application\_error(-20001,'Zmiana ceny nie moze byc wieksza niz 50% wartosci poprzedniej ceny');

RETURN;

END IF;

UPDATE PRODUKTY\_SPOZYWCZE SET CENA\_JEDNOSTKOWA = new\_product\_price

WHERE NAZWA = product\_name;

COMMIT;

dbms\_output.put\_line('Cena produktu zostala zaktualizowana');

END;

#### set\_product\_points

Procedura wykorzystywana do aktualizacji punktów zdobywanych po zakupie danego produktu spożywczego.

CREATE OR REPLACE PROCEDURE set\_product\_points

(product\_name IN VARCHAR2, -- nazwa produktu spożywczego

new\_points IN INTEGER) -- nowa ilość punktów za produkt

IS

product\_exist INT;

BEGIN

IF new\_points < 0 THEN

raise\_application\_error(-20001,'Liczba punktow nie moze byc ujemna');

RETURN;

END IF;

SELECT COUNT(\*) INTO product\_exist FROM produkty\_spozywcze

WHERE NAZWA = product\_name;

IF product\_exist = 0 THEN

raise\_application\_error(-20001,'Nieprawidlowa nazwa produktu');

RETURN;

END IF;

UPDATE Produkty\_spozywcze SET ILOSC\_PUNKTOW = new\_points

WHERE NAZWA = product\_name;

COMMIT;

dbms\_output.put\_line('Ilosc punktow dla danego produktu zostala zaktualizowana');

END;

#### show\_transaction\_analize\_daily\_fuel

Procedura odpowiedzialna za generowanie raportu, w którym możemy analizować dane na temat sprzedaży ilości paliw. W raporcie możemy ujrzeć kolumny odpowiedzialne za: liczbę transakcji przeprowadzonych w danym dniu, łączną kwotę na jakie transakcję zostały wykonane oraz sumę sprzedanego paliwa.

CREATE OR ALTER PROCEDURE show\_transaction\_analize\_daily\_fuel

(@fuel\_id INT = 1) -- numer id paliwa

AS

BEGIN

SELECT data\_transakcji, COUNT(ID\_transakcji) AS NumberOfTransaction,

SUM(kwota\_transakcji) AS SumOfTransaction, SUM(ilosc\_paliwa) SumOfFuel

FROM Transakcje\_paliwowe

WHERE ID\_paliwa = @fuel\_id

GROUP BY data\_transakcji

ORDER BY data\_transakcji DESC;

END;

EXEC show\_transaction\_analize\_daily\_fuel @fuel\_id = 1;

#### show\_transaction\_analize\_products\_daily

Procedura odpowiedzialna za generowanie raportu odnośnie wyników sprzedaży produktów spożywczych. W raporcie możemy ujrzeć dane tj: liczbę przeprowadzonych transakcji w danym dniu, łączną liczbę sprzedanych artykułów spożywczych oraz sumę pieniędzy jaką wygenerowano za sprzedaż tych produktów.

CREATE OR ALTER PROCEDURE show\_transaction\_analize\_products\_daily

(@product\_id INT = 1) -- numer id produktu

AS

BEGIN

SELECT data\_transakcji, COUNT(ID\_transakcji) AS NumberOfTransaction, COUNT(ilosc) AS NumberOfSoldProducts, SUM(kwota\_transakcji) AS SumOfTransaction

FROM Transakcje\_spozywcze

WHERE ID\_produktu = @product\_id

GROUP BY data\_transakcji

ORDER BY data\_transakcji DESC;

END;

EXEC show\_transaction\_analize\_products\_daily @product\_id = 3;

#### show\_history\_of\_fuel\_price

Procedura odpowiedzialna za wyświetlenie informacji o histori cen paliwa, procedura pobierane dane z widoku.

CREATE OR ALTER PROCEDURE show\_history\_of\_fuel\_price(

@fuel\_id INT = 1) -- numer id paliwa

AS

BEGIN

SELECT DATA\_CENY, CENA FROM view\_price\_history

WHERE ID\_TYPU = @fuel\_id

ORDER BY DATA\_CENY DESC;

END;

EXEC dbo.show\_history\_of\_fuel\_price @fuel\_id = 1;

#### show\_orders\_of\_fuel

Procedura odpowiedzialna za wyświetlanie informacji odnośnie złożonych zamówień paliwowych.

CREATE OR ALTER PROCEDURE show\_orders\_of\_fuel(

@fuel\_id INT = 1) -- numer id paliwa

AS

BEGIN

SELECT \*

FROM OPENQUERY(ZaopatrzenieOracle,'SELECT \* FROM show\_order\_fuel\_history')

WHERE ID\_TYPU = @fuel\_id;

END;

EXEC show\_orders\_of\_fuel @fuel\_id = 2;

#### show\_number\_of\_new\_customer

Procedura odpowiedzialna za wyświetlanie informacji odnośnie ilości nowo zarejestrowanych klientów w programie lojalnościowym stacji paliwowej.

CREATE OR ALTER PROCEDURE show\_number\_of\_new\_customer

AS

BEGIN

SELECT YEAR(DATA\_ZALOZENIA\_KONTA) AS 'Year', MONTH(DATA\_ZALOZENIA\_KONTA) AS 'Month',

COUNT(ID\_KLIENTA) AS 'Number Of Customers'

FROM ZaopatrzenieOracle.."ADMINISTRATORORACLE"."KLIENCI"

GROUP BY YEAR(DATA\_ZALOZENIA\_KONTA), MONTH(DATA\_ZALOZENIA\_KONTA)

ORDER BY YEAR(DATA\_ZALOZENIA\_KONTA) DESC, MONTH(DATA\_ZALOZENIA\_KONTA) ASC;

END;

EXEC dbo.show\_number\_of\_new\_customer;

#### update\_petrol\_price

Procedura służąca do aktualizacji cen danego rodzaju paliwa.

create or replace Procedure update\_petrol\_price

(petrol\_name IN varchar2, -- nazwa paliwa

petrol\_price IN NUMBER) -- nowa cena za litr paliwa

IS

petrol\_name\_exist INTEGER;

petrol\_id INTEGER;

new\_price\_id INTEGER;

BEGIN

IF petrol\_price < 0 THEN

raise\_application\_error(-20001,'Cena nie moze byc mniejsza od 0');

RETURN;

END IF;

SELECT COUNT(p.id\_paliwa) INTO petrol\_name\_exist FROM paliwa p

JOIN historia\_cen\_paliw hcp using(id\_ceny)

JOIN typ\_paliwa tp using(id\_typu)

WHERE tp.nazwa\_paliwa = petrol\_name;

IF petrol\_name\_exist = 0 THEN

raise\_application\_error(-20001,'Nieprawidlowa nazwa paliwa');

RETURN;

END IF;

SELECT p.id\_paliwa INTO petrol\_id FROM paliwa p

JOIN historia\_cen\_paliw hcp using(id\_ceny)

JOIN typ\_paliwa tp using(id\_typu)

WHERE tp.nazwa\_paliwa = petrol\_name;

INSERT INTO Historia\_Cen\_Paliw(ID\_typu,data\_ceny,cena) VALUES(petrol\_id,CURRENT\_DATE,petrol\_price);

SELECT ID\_ceny INTO new\_price\_id FROM Historia\_Cen\_Paliw

WHERE ID\_ceny = (SELECT MAX(ID\_ceny) FROM Historia\_Cen\_Paliw);

UPDATE Paliwa SET ID\_ceny = new\_price\_id WHERE ID\_paliwa = petrol\_id;

COMMIT;

dbms\_output.put\_line('Cena Paliwa zostala zaktualizowana');

END;

#### update\_petrol\_points

Procedura służąca do aktualizowania ilości punktów zdobywanych za dany rodzaj paliwa.

CREATE OR REPLACE Procedure update\_petrol\_points

(petrol\_name IN varchar2, -- nazwa paliwa

petrol\_points IN NUMBER) -- nowa liczba punktów przyznawana przy sprzedaży

IS

petrol\_name\_exist INTEGER;

petrol\_id INTEGER;

new\_points\_id INTEGER;

BEGIN

IF petrol\_points < 0 THEN

raise\_application\_error(-20001,'Ilosc punktow nie moze byc mniejsza od 0');

RETURN;

END IF;

SELECT COUNT(p.id\_paliwa) INTO petrol\_name\_exist FROM paliwa p

JOIN historia\_cen\_paliw hcp using(id\_ceny)

JOIN typ\_paliwa tp using(id\_typu)

WHERE tp.nazwa\_paliwa = petrol\_name;

IF petrol\_name\_exist = 0 THEN

raise\_application\_error(-20001,'Nieprawidlowa nazwa paliwa');

RETURN;

END IF;

SELECT p.id\_paliwa INTO petrol\_id FROM paliwa p

JOIN historia\_cen\_paliw hcp using(id\_ceny)

JOIN typ\_paliwa tp using(id\_typu)

WHERE tp.nazwa\_paliwa = petrol\_name;

UPDATE Paliwa SET ilosc\_punktow = petrol\_points

WHERE ID\_paliwa = petrol\_id;

COMMIT;

dbms\_output.put\_line('Punkty dla danego paliwa zostaly zaktualizowane');

END;

#### show\_statistic\_car\_wash

Procedura odpowiedzialna za wyświetlenie informacji o statystykach związanych z myjnią samochodową.

CREATE OR ALTER PROCEDURE show\_statistic\_car\_wash

AS

BEGIN

SELECT NAZWA\_PROGRAMU, COUNT(NAZWA\_PROGRAMU) AS NumberOfTransaction, DATA\_TRANSAKCJI

FROM OPENQUERY(ZaopatrzenieOracle,'SELECT \* FROM MYJNIA.ROZSZERZONE\_TRANSAKCJE\_MYJNI')

GROUP BY NAZWA\_PROGRAMU, DATA\_TRANSAKCJI

ORDER BY DATA\_TRANSAKCJI DESC, NAZWA\_PROGRAMU ASC;

END;

GO

EXEC show\_statistic\_car\_wash;

#### show\_employee\_hours

Procedura odpowiedzialna za wyświetlanie informacji odnośnie przepracowanych godzin oraz przewidywanych zarobków przez danego pracownika za dany okres czasu.

CREATE OR ALTER PROCEDURE show\_employee\_hours

(@emp\_id INT = 1) -- numer id pracownika

AS

BEGIN

SELECT Prac.imie, Prac.Nazwisko,

SUM(DATEDIFF(HOUR,data\_rozpoczecia\_zmiany,data\_zakonczenia\_zmiany)) AS TotalHourWork,

SUM(DATEDIFF(HOUR,data\_rozpoczecia\_zmiany,data\_zakonczenia\_zmiany)) \* Prac.[pensja za godzinę] AS EstimatedSalary,

YEAR(data\_rozpoczecia\_zmiany) AS 'YEAR', MONTH(data\_rozpoczecia\_zmiany) AS 'Month'

FROM Harmonogram AS Harm

JOIN Pracownicy...[Pracownicy$] AS Prac

ON Prac.Id\_pracownika = Harm.ID\_pracownika

WHERE Harm.ID\_pracownika = @emp\_id

GROUP BY Prac.imie, Prac.Nazwisko, Prac.ID\_pracownika,Prac.[pensja za godzinę],

YEAR(data\_rozpoczecia\_zmiany), MONTH(data\_rozpoczecia\_zmiany);

END;

#### show\_employee\_works\_all

Procedura odpowiedzialna za wyświetlenie informacji odnośnie wydatków związanych z wynagrodzeniami dla pracowników stacji paliw za dany okres pracy.

CREATE OR ALTER PROCEDURE show\_employee\_works\_all

AS

BEGIN

SELECT temp.Year\_, temp.Month\_, SUM(temp.EstimatedSalary) AS TotalPaidPerMonth

FROM (

SELECT YEAR(data\_rozpoczecia\_zmiany) AS Year\_, MONTH(data\_rozpoczecia\_zmiany) AS Month\_,

SUM(DATEDIFF(HOUR,data\_rozpoczecia\_zmiany,data\_zakonczenia\_zmiany)) \* Prac.[pensja za godzinę] AS EstimatedSalary

FROM Harmonogram AS Harm

JOIN Pracownicy...[Pracownicy$] AS Prac

ON Prac.Id\_pracownika = Harm.ID\_pracownika

GROUP BY Prac.[pensja za godzinę],

YEAR(data\_rozpoczecia\_zmiany), MONTH(data\_rozpoczecia\_zmiany)) AS temp

GROUP BY temp.Year\_, temp.Month\_;

END;

EXEC show\_employee\_works\_all;

### Widoki

#### view\_price\_history

Widok wyświetlający informację o historii cen paliw, które zostają pobrane z serwera Oracle.

CREATE OR ALTER VIEW view\_price\_history

AS

SELECT ID\_TYPU, DATA\_CENY, CENA FROM OPENQUERY(ZaopatrzenieOracle,'SELECT \* FROM ADMINISTRATORORACLE.historia\_cen\_paliw');

# Replikacja danych

Wykorzystanie replikacji danych pozwala min. Na zwiększenie dostępności danych oraz na bezpieczeństwo, ponieważ przechowujemy dane w więcej niż jednym miejscu.

Replikacja danych następuje pomiędzy dwoma bazami danych: StacjaPaliw oraz StacjaZarząd. W ten sposób zarząd prowadzi analizę oraz statystyki na zreplikowanych danych, dzięki czemu odciąża główną bazę danych obsługującą stację paliw.

Aby wykorzystać mechanizm replikacji należy wykonać odpowiednią konfigurację serwera SQL. Następnym krokiem będzie utworzenie publikacji, która będzie zawierała artykuły, który będą replikowane na inny serwer.

use master

exec sp\_adddistributor @distributor = N'Miłosz', @password = N''

GO

exec sp\_adddistributiondb @database = N'distribution', @data\_folder = N'E:\MS\_SQL\_Server\MSSQL15.MSSQLSERVER\MSSQL\Data', @log\_folder = N'E:\MS\_SQL\_Server\MSSQL15.MSSQLSERVER\MSSQL\Data', @log\_file\_size = 2, @min\_distretention = 0, @max\_distretention = 72, @history\_retention = 48, @deletebatchsize\_xact = 5000, @deletebatchsize\_cmd = 2000, @security\_mode = 1

GO

use [distribution]

if (not exists (select \* from sysobjects where name = 'UIProperties' and type = 'U '))

create table UIProperties(id int)

if (exists (select \* from ::fn\_listextendedproperty('SnapshotFolder', 'user', 'dbo', 'table', 'UIProperties', null, null)))

EXEC sp\_updateextendedproperty N'SnapshotFolder', N'E:\MS\_SQL\_Server\MSSQL15.MSSQLSERVER\MSSQL\ReplData', 'user', dbo, 'table', 'UIProperties'

else

EXEC sp\_addextendedproperty N'SnapshotFolder', N'E:\MS\_SQL\_Server\MSSQL15.MSSQLSERVER\MSSQL\ReplData', 'user', dbo, 'table', 'UIProperties'

GO

exec sp\_adddistpublisher @publisher = N'MIŁOSZ', @distribution\_db = N'distribution', @security\_mode = 0, @login = N'sa', @password = N'', @working\_directory = N'E:\MS\_SQL\_Server\MSSQL15.MSSQLSERVER\MSSQL\ReplData', @trusted = N'false', @thirdparty\_flag = 0, @publisher\_type = N'MSSQLSERVER'

GO

use [StacjaPaliw]

exec sp\_replicationdboption @dbname = N'StacjaPaliw', @optname = N'publish', @value = N'true'

GO

-- Adding the snapshot publication

use [StacjaPaliw]

exec sp\_addpublication @publication = N'StacjaPaliw\_transakcje', @description = N'Snapshot publication of database ''StacjaPaliw'' from Publisher ''Miłosz''.', @sync\_method = N'native', @retention = 0, @allow\_push = N'true', @allow\_pull = N'true', @allow\_anonymous = N'true', @enabled\_for\_internet = N'false', @snapshot\_in\_defaultfolder = N'true', @compress\_snapshot = N'false', @ftp\_port = 21, @ftp\_login = N'anonymous', @allow\_subscription\_copy = N'false', @add\_to\_active\_directory = N'false', @repl\_freq = N'snapshot', @status = N'active', @independent\_agent = N'true', @immediate\_sync = N'true', @allow\_sync\_tran = N'false', @autogen\_sync\_procs = N'false', @allow\_queued\_tran = N'false', @allow\_dts = N'false', @replicate\_ddl = 1

GO

exec sp\_addpublication\_snapshot @publication = N'StacjaPaliw\_transakcje', @frequency\_type = 4, @frequency\_interval = 1, @frequency\_relative\_interval = 1, @frequency\_recurrence\_factor = 0, @frequency\_subday = 4, @frequency\_subday\_interval = 2, @active\_start\_time\_of\_day = 0, @active\_end\_time\_of\_day = 235959, @active\_start\_date = 0, @active\_end\_date = 0, @job\_login = null, @job\_password = null, @publisher\_security\_mode = 1

use [StacjaPaliw]

exec sp\_addarticle @publication = N'StacjaPaliw\_transakcje', @article = N'Harmonogram', @source\_owner = N'dbo', @source\_object = N'Harmonogram', @type = N'logbased', @description = null, @creation\_script = null, @pre\_creation\_cmd = N'drop', @schema\_option = 0x000000000803509D, @identityrangemanagementoption = N'manual', @destination\_table = N'Harmonogram', @destination\_owner = N'dbo', @vertical\_partition = N'false'

GO

use [StacjaPaliw]

exec sp\_addarticle @publication = N'StacjaPaliw\_transakcje', @article = N'Transakcje\_paliwowe', @source\_owner = N'dbo', @source\_object = N'Transakcje\_paliwowe', @type = N'logbased', @description = null, @creation\_script = null, @pre\_creation\_cmd = N'drop', @schema\_option = 0x000000000803509D, @identityrangemanagementoption = N'manual', @destination\_table = N'Transakcje\_paliwowe', @destination\_owner = N'dbo', @vertical\_partition = N'false'

GO

use [StacjaPaliw]

exec sp\_addarticle @publication = N'StacjaPaliw\_transakcje', @article = N'Transakcje\_spozywcze', @source\_owner = N'dbo', @source\_object = N'Transakcje\_spozywcze', @type = N'logbased', @description = null, @creation\_script = null, @pre\_creation\_cmd = N'drop', @schema\_option = 0x000000000803509D, @identityrangemanagementoption = N'manual', @destination\_table = N'Transakcje\_spozywcze', @destination\_owner = N'dbo', @vertical\_partition = N'false'

GO

use [StacjaPaliw]

exec sp\_replicationdboption @dbname = N'StacjaPaliw', @optname = N'publish', @value = N'true'

GO

Następnym krokiem będzie utworzenie subskrypcji dla bazy danych StacjaZarzad:

use [StacjaZarzad]

exec sp\_addsubscription @publication = N'StacjaPaliw\_transakcje', @subscriber = N'MIŁOSZ', @destination\_db = N'StacjaZarzad', @subscription\_type = N'Push', @sync\_type = N'automatic', @article = N'all', @update\_mode = N'read only', @subscriber\_type = 0

exec sp\_addpushsubscription\_agent @publication = N'StacjaPaliw\_transakcje', @subscriber = N'MIŁOSZ', @subscriber\_db = N'StacjaZarzad', @job\_login = null, @job\_password = null, @subscriber\_security\_mode = 1, @frequency\_type = 64, @frequency\_interval = 0, @frequency\_relative\_interval = 0, @frequency\_recurrence\_factor = 0, @frequency\_subday = 0, @frequency\_subday\_interval = 0, @active\_start\_time\_of\_day = 0, @active\_end\_time\_of\_day = 235959, @active\_start\_date = 20240621, @active\_end\_date = 99991231, @enabled\_for\_syncmgr = N'False', @dts\_package\_location = N'Distributor'

Po odpowiednim utworzeniu publikacji oraz subskrypcji możemy sprawdzić poprawność replikacji, wstawiając dane w replikowanej tabeli w bazie danych StacjaPaliw oraz pobraniu danych z tej samej tabeli ale juz w bazie danych: StacjaZarzad.

# Transakcje rozproszone

Transakcje rozproszone to transakcje, które obejmują operacje wykonywane na więcej niż jednym zasobie zarządzanym przez różne systemy zarządzania bazą danych (DBMS) lub inne zasoby transakcyjne. Przykładem może być operacja, która musi zaktualizować dane w dwóch różnych bazach danych znajdujących się na różnych serwerach.

Transakcje rozproszone działają na zasadzie protokołu dwufazowego zatwierdzania (2PC - Two-Phase Commit Protocol):

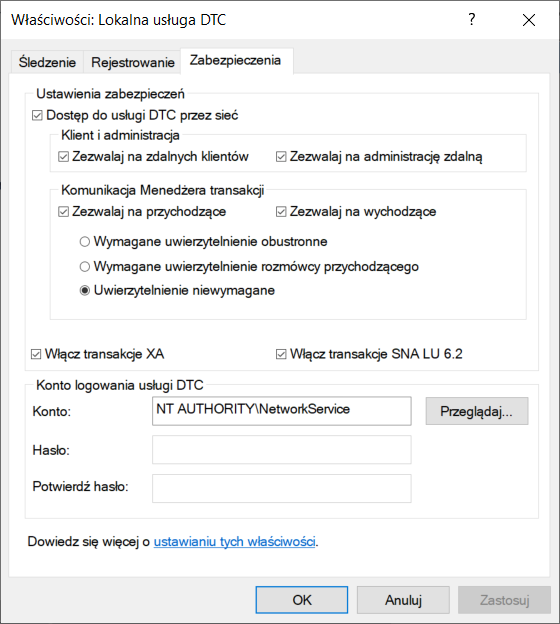
1. **Faza przygotowania (Prepare Phase):**
   * Koordynator transakcji (w tym przypadku MS DTC) wysyła żądanie do wszystkich uczestniczących zasobów (np. bazy danych) z prośbą o przygotowanie transakcji.
   * Każdy zasób wykonuje operacje transakcyjne lokalnie i przygotowuje się do zatwierdzenia transakcji, ale jej nie zatwierdza. Jeśli zasób jest gotowy do zatwierdzenia, wysyła odpowiedź „przygotowany” do koordynatora.
   * Jeśli którykolwiek z zasobów zgłosi błąd lub nie może przygotować transakcji, wysyła odpowiedź „niepowodzenie” do koordynatora.
2. **Faza zatwierdzenia (Commit Phase):**
   * Jeśli wszystkie zasoby zgłosiły, że są przygotowane, koordynator transakcji wysyła polecenie zatwierdzenia do wszystkich zasobów.
   * Każdy zasób zatwierdza transakcję.
   * Jeśli którykolwiek z zasobów zgłosił niepowodzenie w fazie przygotowania, koordynator wysyła polecenie wycofania do wszystkich zasobów, które wycofują transakcję.

## Konfiguracja MS DTC

Microsoft Distributed Transaction Coordinator (MS DTC) jest usługą, która zarządza transakcjami rozproszonymi w systemie Windows.

Pierwszym krokiem konfiguracji MS DTC jest uruchomienie usługi: Kordynatora Transakcji Rozproszonych.

Następnie przechodzimy do konfiguracji MS DTC za pomocą odpowiedniej przystawki:



W zakładce Zabezpieczenia zaznaczamy odpowiednie opcje.

Kolejnym krokiem jest odpowiednie skonfigurowanie zapory sieciowej oraz portów, które są wykorzystywane przez MS DTC (domyślnie port 135 oraz dynamiczne porty RPC).

Finalnie restartujemy usługę MS DTC, aby mieć pewność, że wprowadzone przez nas zmiany konfiguracji zostały przyjęte i wprowadzone.

Przykładowa procedura wykorzystująca transakcje rozproszone:

CREATE OR ALTER PROCEDURE delete\_client

(

@name VARCHAR(40), -- nazwa klienta

@surname VARCHAR(50), -- nazwisko klienta

@id INT -- ID klienta

)

AS

DECLARE @sql NVARCHAR(MAX);

BEGIN

IF @id IS NOT NULL

BEGIN

DECLARE @client\_exists INT;

SELECT @client\_exists = COUNT(\*) FROM ZaopatrzenieOracle.."ADMINISTRATORORACLE"."KLIENCI" WHERE ID\_klienta = @id;

IF @client\_exists < 1

BEGIN

RAISERROR('Cannot find client to delete', 16, 1);

END

SET @sql = N'DELETE FROM ADMINISTRATORORACLE.KLIENCI WHERE ID\_klienta = ' + CAST(@id AS VARCHAR);

EXEC (@sql) AT ZaopatrzenieOracle;

BEGIN DISTRIBUTED TRANSACTION;

UPDATE Transakcje\_paliwowe SET ID\_klienta = NULL WHERE ID\_klienta = @id;

UPDATE Transakcje\_spozywcze SET ID\_klienta = NULL WHERE ID\_klienta = @id;

COMMIT TRANSACTION;

END

ELSE

BEGIN

DECLARE @client\_by\_name\_id INT;

IF @name IS NULL OR @surname IS NULL

BEGIN

RAISERROR('Cannot find client to delete', 16, 1);

END

SELECT @client\_by\_name\_id = ID\_klienta FROM ZaopatrzenieOracle.."ADMINISTRATORORACLE"."KLIENCI" WHERE imie = @name AND nazwisko = @surname

IF @client\_by\_name\_id IS NULL

BEGIN

RAISERROR('Cannot find client to delete', 16, 1);

END

SET @sql = N'DELETE FROM ADMINISTRATORORACLE.KLIENCI WHERE imie = ''' + CAST(@name AS VARCHAR(30)) + ''' AND nazwisko = ''' + CAST(@surname AS VARCHAR(50)) + '''';

EXEC (@sql) AT ZaopatrzenieOracle;

BEGIN DISTRIBUTED TRANSACTION

UPDATE Transakcje\_paliwowe SET ID\_klienta = NULL WHERE ID\_klienta = @client\_by\_name\_id;

UPDATE Transakcje\_spozywcze SET ID\_klienta = NULL WHERE ID\_klienta = @client\_by\_name\_id;

COMMIT TRANSACTION;

END END;

# Triggery

## Rozszerzone\_Transakcje\_Myjni\_trigger

Trigger wykorzystywany do wprowadzania danych do widoku Rozszerzone\_Transakcje\_Myjni

CREATE OR REPLACE TRIGGER Rozszerzone\_Transakcje\_Myjni\_trigger

INSTEAD OF INSERT ON Rozszerzone\_Transakcje\_Myjni

FOR EACH ROW

DECLARE

program\_exist INT;

client\_exist INT;

BEGIN

--sprawdzenie czy istnieje dany program myjni

SELECT COUNT(\*) INTO program\_exist FROM programy\_mycia WHERE nazwa=:NEW.NAZWA\_PROGRAMU;

IF program\_exist = 0 THEN

raise\_application\_error(-20001,'Program mycia nie istnieje w bazie danych');

RETURN;

END IF;

--dodanie transakcji dla myjni (klient istnieje w bazie lub nie)

SELECT COUNT(\*) INTO client\_exist FROM klienci@StacjaPubliczny WHERE IMIE = :NEW.IMIE AND NAZWISKO = :NEW.NAZWISKO;

IF client\_exist = 0 THEN

INSERT INTO TRANSAKCJE\_MYJNI(ID\_PROGRAMU,ID\_KLIENTA,DATA\_TRANSAKCJI)

VALUES ((SELECT ID\_PROGRAMU FROM PROGRAMY\_MYCIA WHERE NAZWA = :NEW.NAZWA\_PROGRAMU FETCH FIRST 1 ROW ONLY),NULL,SYSDATE);

ELSE

INSERT INTO TRANSAKCJE\_MYJNI(ID\_PROGRAMU,ID\_KLIENTA,DATA\_TRANSAKCJI)

VALUES ((SELECT ID\_PROGRAMU FROM PROGRAMY\_MYCIA WHERE NAZWA = :NEW.NAZWA\_PROGRAMU FETCH FIRST 1 ROW ONLY),

(SELECT ID\_KLIENTA FROM KLIENCI@StacjaPubliczny WHERE IMIE=:NEW.IMIE AND NAZWISKO = :NEW.NAZWISKO FETCH FIRST 1 ROW ONLY),SYSDATE);

END IF;

END;

* on\_petrol\_delivery

Trigger odpowiedzialny za sprawdzenie czy nie ma już dostawy paliwa danego dnia.

CREATE OR REPLACE TRIGGER ON\_PETROL\_DELIVERY

BEFORE INSERT ON ZAMOWIENIA\_PALIWOWE

FOR EACH ROW

DECLARE delivery\_present INTEGER;

BEGIN

SELECT COUNT(\*)

INTO delivery\_present

FROM ZAMOWIENIA\_PALIWOWE

WHERE EXTRACT(YEAR FROM DATA\_ZAMOWIENIA) = EXTRACT(YEAR FROM :NEW.DATA\_ZAMOWIENIA)

AND EXTRACT(MONTH FROM DATA\_ZAMOWIENIA) = EXTRACT(MONTH FROM :NEW.DATA\_ZAMOWIENIA)

AND EXTRACT(DAY FROM DATA\_ZAMOWIENIA) = EXTRACT(DAY FROM :NEW.DATA\_ZAMOWIENIA);

IF delivery\_present > 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Tego dnia paliwo juz bylo zamawiane');

END IF;

END;

* on\_products\_delivery

Trigger odpowiedzialny za sprawdzenie czy nie ma już dostawy produktów spożywczych danego dnia

CREATE OR REPLACE TRIGGER ON\_PRODUCTS\_DELIVERY

BEFORE INSERT ON ZAMOWIENIA\_SPOZYWCZE

FOR EACH ROW

DECLARE

delivery\_present INTEGER;

BEGIN

SELECT COUNT(\*)

INTO delivery\_present

FROM ZAMOWIENIA\_SPOZYWCZE

WHERE EXTRACT(YEAR FROM DATA\_ZAMOWIENIA) = EXTRACT(YEAR FROM :NEW.DATA\_ZAMOWIENIA)

AND EXTRACT(MONTH FROM DATA\_ZAMOWIENIA) = EXTRACT(MONTH FROM :NEW.DATA\_ZAMOWIENIA)

AND EXTRACT(DAY FROM DATA\_ZAMOWIENIA) = EXTRACT(DAY FROM :NEW.DATA\_ZAMOWIENIA);

IF delivery\_present > 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Tego dnia produkty spozywcze juz bylo zamawiane');

END IF;

END;