Projekt Bazy Danych

Wojciech Jasiński i Krystian Włodek

Wyróżnione role w systemie

System przewiduje pięć rodzajów użytkowników:

- Bez konta
- Klient (indywidualny lub firma)
- Pracownik (obsługujący zamówienia)
- Menedżer
- Administrator

Opisy ról:

- **Użytkownik bez konta** może:
    1. zarejestrować się
    2. przeglądać aktualne menu restauracji
    3. przeglądać dostępne w danym okresie czasu dostępność stolików
    4. przeglądać możliwe rabaty
    5. wykonać zamówienie ale jedynie telefonicznie

- **Klient** może:
    1. zalogować się
    2. zresetować hasło poprzez podany adres e-mail
    3. zmieniać dane konta
    4. przeglądać aktualne menu restauracji
    5. przeglądać dostępne w danym okresie czasu dostępność stolików
    6. przeglądać możliwe rabaty (tylko indywidualny)
    7. wykonać zamówienie (zarówno prywatne jak i firmowe) telefonicznie lub poprzez dostępny formularz, równocześnie decydując czy na miejscu, czy na wynos. W przypadku zamówienia na miejscu, przechodzi do formularza rezerwacji stolika.
    8. przeglądać historię swoich zamówień i swoje faktury (jeżeli jakieś posiada)
    9. usunąć konto

- **Pracownik** może:
    1. zalogować się
    2. zresetować hasło poprzez podany adres e-mail
    3. zmieniać dane konta
    4. przeglądać aktualne menu restauracji
    5. przeglądać zaplanowane rezerwacje stolików

6. przeglądać możliwe rabaty
7. przeglądać historię zamówień i faktury klientów (jeżeli jakieś posiadają)
8. odbierać zamówienia oraz je modyfikować (zarówno od użytkowników bez konta jak i z kontem) i wprowadzać je do systemu
9. zakładać oraz dezaktywować konta klientom restauracji

- **Menedżer** może:
    1. zalogować się
    2. zresetować hasło poprzez podany adres e-mail
    3. zmieniać dane konta
    4. przeglądać aktualne (oraz planować przyszłe) menu restauracji
    5. przeglądać i modyfikować zaplanowane rezerwacje stolików
    6. przeglądać możliwe rabaty
    7. przeglądać historię zamówień i faktury klientów (jeżeli jakieś posiadają)
    8. generować raporty zbiorcze
    9. odbierać zamówienia oraz je modyfikować (zarówno od użytkowników bez konta jak i z kontem) i wprowadzać je do systemu
    10. zakładać oraz dezaktywować konta klientom restauracji
    11. zakładać oraz dezaktywować konta pracownikom restauracji
    12. generować raporty zbiorcze całej restauracji

- **Administrator** może:
    1. zalogować się
    2. zresetować hasło poprzez podany adres e-mail
    3. zmieniać dane konta
    4. przeglądać oraz modyfikować aktualne menu restauracji
    5. przeglądać oraz modyfikować zaplanowane rezerwacje stolików
    6. przeglądać oraz modyfikować możliwe rabaty
    7. przeglądać historię zamówień i faktury klientów (jeżeli jakieś posiadają)
    8. generować raporty zbiorcze na życzenie klienta
    9. odbierać zamówienia oraz je modyfikować (zarówno od użytkowników bez konta jak i z kontem) i wprowadzać je do systemu
    10. zakładać konta oraz dezaktywować klientom restauracji
    11. zakładać oraz dezaktywować konta pracownikom restauracji
    12. zakładać i dezaktywować konta menedżerom restauracji
    13. generować raporty zbiorcze całej restauracji

Konta pracowników, menedżerów oraz administratorów nie zawierają funkcjonalności konta klienta. Chcąc złożyć zamówienie powinni oni korzystać z prywatnego konta klienta.

Uprawnienia konta pracownika obejmują te potrzebne do wykonywania codziennych obowiązków.
Uprawnienia konta menedżera, oprócz uprawnień zwykłego pracownika, zawierają funkcje potrzebne/przydatne do kierowania restauracją.
Konto administratora posiada wszystkie możliwe uprawnienia systemu.

Zamówienia i rezerwacje

Użytkownik bez konta ma możliwość złożenia zamówienia telefonicznie lub na miejscu bez potrzeby zakładania konta (również z wyprzedzeniem i rezerwacją).

W przypadku braku rezerwacji i braku stolików, klient musi poczekać aż jakiś się zwolni lub sobie pójść.

Klient ponadto, ma możliwość skorzystania z aplikacji i w niej złożyć zamówienie (wraz z rezerwacją) zarówno firmowe, indywidualne jak i na “catering” z odbiorem osobistym. Ma dostęp również do całej historii zamówień jak i wszystkich dowodów zakupu (np. faktura).

Pracownik obsługuje zamówienia, ma wgląd w informacje potrzebne do realizowania swoich obowiązków.

Menedżer posiada dodatkowe, wymagające większej odpowiedzialności uprawnienia, by móc rozwiązywać proste problemy w restauracji na bieżąco.

Administrator posiada wszystkie uprawnienia systemu (również te niebezpieczne), by móc reagować natychmiast na zgłoszone błędy itp.

- **Klient** może:
    1. złożyć zamówienie z odbiorem na miejscu lub na wynos
    2. złożyć zamówienie z rezerwacją stolika w restauracji
    3. zmodyfikować lub anulować swoje zamówienie (również z rezerwacją). W przypadku, gdy klient anuluje je za późno musi opłacić koszty realizacji takiego zamówienia.
    4. przeglądać status swojego aktualnego/planowanego zamówienia
    5. przeglądać historię swoich zamówień i dowodów zakupu
    6. przeglądać status swoich rabatów (tylko indywidualny)

- **Pracownik** może:
    1. potwierdzić lub odrzucić (powinien skontaktować się z klientem telefonicznie) zamówienie
    2. złożyć zamówienie na miejscu lub na wynos dla klienta
    3. odebrać zamówienie i przydzielić do stolika klienta na miejscu (bez rezerwacji)
    4. przeglądać aktualne i planowane zamówienia
    5. przeglądać historię zamówień na życzenie klienta

- **Menedżer** może:

1. potwierdzić lub odrzucić (powinien skontaktować się z klientem telefonicznie) zamówienie
2. złożyć zamówienie na miejscu lub na wynos dla klienta
3. odebrać zamówienie i przydzielić do stolika klienta na miejscu (bez rezerwacji)
4. przeglądać aktualne i planowane zamówienia
5. przeglądać historię wszystkich zamówień restauracji
6. anulować zamówienie
7. przeglądać historię zamówień i rezerwacji
8. przeglądać szczegóły kont klientów

Menu

Menu posiada pozycje, które można aktywować lub dezaktywować - w ten sposób menu może cały czas się zmieniać, a pozycje cały czas są w systemie, niewidoczne dla klienta.

- **Klient** może:
    1. Zobaczyć wszystkie aktywne pozycje w menu

- **Pracownik** może:
    1. Zobaczyć wszystkie aktywne i nieaktywne pozycje w menu oraz całą historię zmian w menu

- **Menedżer** może:
    1. Zobaczyć wszystkie aktywne i nieaktywne pozycje w menu oraz całą historię zmian w menu
    2. zaplanować aktywację pozycji w menu
    3. zaplanować dezaktywację pozycji w menu

- **Administrator** może:
    1. Zobaczyć wszystkie aktywne i nieaktywne pozycje w menu oraz całą historię zmian w menu
    2. zaplanować aktywację pozycji w menu
    3. zaplanować dezaktywację pozycji w menu
    4. Dodać nową pozycję w menu
    5. Usunąć pozycję z menu
    6. Edytować pozycję w menu

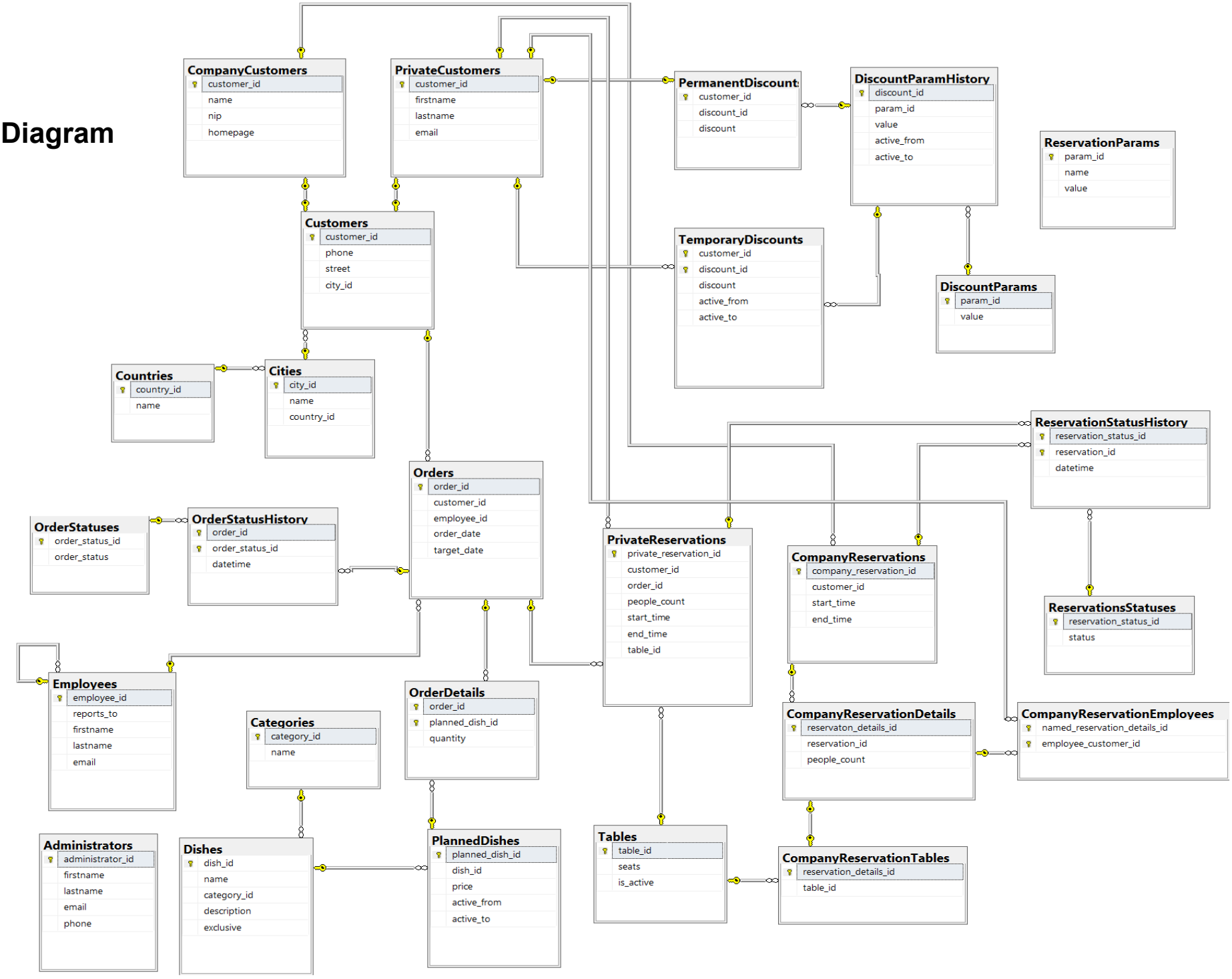
Inne

- **Menedżer** może:
    1. aktywować/dezaktywować stolik
    2. Generować raporty dotyczące:
        a. stolików

    b. rabatów
    c. menu
    d. statystyk zamówień (kwota, czas składania) dla klienta indywidualnego lub firm

- **Administrator** może:
    1. aktywować/dezaktywować stolik
    2. Generować raporty dotyczące:
        a. stolików
        b. rabatów
        c. menu
        d. statystyk zamówień (kwota, czas składania) dla klienta indywidualnego lub firm
    3. Zmienić reguły przyznawania rabatów
    4. dodać/usunąć stolik

Diagram


CompanyCustomers
customer_id
name
nip
homepage
PrivateCustomers
customer_id
firstname
lastname
email
PermanentDiscount
customer_id
discount_id
discount
DiscountParamHistory
discount_id
param_id
value
active_from
active_to
ReservationParams
param_id
name
value
Customers
customer_id
phone
street
city_id
TemporaryDiscounts
customer_id
discount_id
discount
active_from
active_to
DiscountParams
param_id
value
Countries
country_id
name
Cities
city_id
name
country_id
ReservationStatusHistory
reservation_status_id
reservation_id
datetime
Orders
order_id
customer_id
employee_id
order_date
target_date
OrderStatuses
order_status_id
order_status
OrderStatusHistory
order_id
order_status_id
datetime
PrivateReservations
private_reservation_id
customer_id
order_id
people_count
start_time
end_time
table_id
CompanyReservations
company_reservation_id
customer_id
start_time
end_time
ReservationsStatuses
reservation_status_id
status
Employees
employee_id
reports_to
firstname
lastname
email
OrderDetails
order_id
planned_dish_id
quantity
Categories
category_id
name
CompanyReservationDetails
reservaton_details_id
reservation_id
people_count
CompanyReservationEmployees
named_reservation_details_id
employee_customer_id
Administrators
administrator_id
firstname
lastname
email
phone
Dishes
dish_id
name
category_id
description
exclusive
PlannedDishes
planned_dish_id
dish_id
price
active_from
active_to
Tables
table_id
seats
is_active
CompanyReservationTables
reservation_details_id
table_id

Tabele

Każde pole tabeli domyślnie jest ustawione na NOT NULL, chyba że napisane jest inaczej

Administrators

Tabela przechowująca dane administratorów.

- **PK** administrator_id **int** - ID administratora
- firstname **nvarchar(50)** - imię administratora
- lastname **nvarchar(50)** - nazwisko administratora
- email **varchar(50)** - adres email administratora
- phone **varchar(15)** - nr telefonu administratora

Warunki integralnościowe:

- email musi być postaci sth@example.ex:
  `CONSTRAINT [CK_email] CHECK  ([email] like '%@%[.]%')`
- phone musi składać się z samych cyfr:
  `CONSTRAINT [CK_phone] CHECK  ([phone] not like '%[^0-9]%')`

```
CREATE TABLE [dbo].[Administrators](
	[administrator_id] [int] IDENTITY(1,1) NOT NULL,
	[firstname] [nvarchar](50) NOT NULL,
	[lastname] [nvarchar](50) NOT NULL,
	[email] [varchar](50) NOT NULL,
	[phone] [varchar](15) NOT NULL,
 CONSTRAINT [PK_Administrators] PRIMARY KEY CLUSTERED
(
	[administrator_id] ASC
),
CONSTRAINT [unique_email] UNIQUE NONCLUSTERED
(
	[email] ASC
)
)
GO

ALTER TABLE [dbo].[Administrators]  WITH CHECK ADD  CONSTRAINT
[CK_email] CHECK  ([email] like '%@%[.]%')
GO
ALTER TABLE [dbo].[Administrators] CHECK CONSTRAINT [CK_email]
```

```
GO

ALTER TABLE [dbo].[Administrators]  WITH CHECK ADD  CONSTRAINT
[CK_phone] CHECK  ([phone] not like '%[^0-9]%')
GO
ALTER TABLE [dbo].[Administrators] CHECK CONSTRAINT [CK_phone]
GO
```

Categories

Słownik dla kategorii dań.

- **PK** category_id **int** - ID kategorii
- name **nvarchar(50)** - nazwa kategorii

Warunki integralnościowe:

- name jest unikalne
  `CONSTRAINT [unique_name] UNIQUE NONCLUSTERED`

```
CREATE TABLE [dbo].[Categories](
	[category_id] [int] IDENTITY(1,1) NOT NULL,
	[name] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_Categories] PRIMARY KEY CLUSTERED
(
	[category_id] ASC
),
CONSTRAINT [unique_name] UNIQUE NONCLUSTERED
(
	[name] ASC
)
)
GO
```

Cities

Słownik dla miast.

- **PK** city_id **int** - ID miasta
- name **nvarchar(50)** - nazwa miasta
- **FK** country_id **nvarchar(50)** - ID kraju, w którym miasto się znajduje

Warunki integralnościowe:

- name jest unikalne
  `CONSTRAINT [unique_name] UNIQUE NONCLUSTERED`

```
CREATE TABLE [dbo].[Cities](
	[city_id] [int] IDENTITY(1,1) NOT NULL,
	[name] [nvarchar](50) NOT NULL,
	[country_id] [int] NOT NULL,
 CONSTRAINT [PK_Cities] PRIMARY KEY CLUSTERED
(
	[city_id] ASC
)
CONSTRAINT [unique_name] UNIQUE NONCLUSTERED
(
	[name] ASC
)
)
GO

ALTER TABLE [dbo].[Cities]  WITH CHECK ADD  CONSTRAINT
[FK_Cities_Countries] FOREIGN KEY([country_id])
REFERENCES [dbo].[Countries] ([country_id])
GO
ALTER TABLE [dbo].[Cities] CHECK CONSTRAINT [FK_Cities_Countries]
GO
```

CompanyCustomers

Tabela z danymi firmowych klientów.

- **PK** customer_id **int** - ID klienta
- name **nvarchar(50)** - nazwa firmy
- nip **varchar(50)** - numer identyfikacji podatkowej
- homepage **varchar(50)** NULL - strona internetowa firmy

Warunki integralnościowe:

- nip jest unikalne
  `CONSTRAINT [unique_nip] UNIQUE NONCLUSTERED`

```
CREATE TABLE [dbo].[CompanyCustomers](
	[customer_id] [int] NOT NULL,
	[name] [nvarchar](50) NOT NULL,
	[nip] [varchar](50) NOT NULL,
	[homepage] [varchar](50) NULL,
 CONSTRAINT [PK_CompanyCustomers] PRIMARY KEY CLUSTERED
(
	[customer_id] ASC
)
CONSTRAINT [unique_nip] UNIQUE NONCLUSTERED
(
	[nip] ASC
)
)
GO

ALTER TABLE [dbo].[CompanyCustomers]  WITH CHECK ADD  CONSTRAINT
[FK_CompanyCustomers_Customers] FOREIGN KEY([customer_id])
REFERENCES [dbo].[Customers] ([customer_id])
GO
ALTER TABLE [dbo].[CompanyCustomers] CHECK CONSTRAINT
[FK_CompanyCustomers_Customers]
GO
```

CompanyReservationDetails

Zawiera szczegółowe informacje dla rezerwacji firmowej. Część rezerwacji, którą opisuje, może mieć przypisany jeden stolik.

- **PK** reservation_details_id **int** - ID
- **FK** reservation_id **int** - ID rezerwacji, której szczegóły dotyczą
- people_count **int** - podana przez klienta ilość osób, dla której ma być przydzielony stolik

Warunki integralnościowe:

- people_count musi być większe od 1:
  `CONSTRAINT [CK_positive] CHECK  ([people_count]>(1))`

```
CREATE TABLE [dbo].[CompanyReservationDetails](
      [reservaton_details_id] IDENTITY(1,1) [int] NOT NULL,
      [reservation_id] [int] NOT NULL,
      [people_count] [int] NOT NULL,
 CONSTRAINT [PK_CompanyReservationDetails] PRIMARY KEY CLUSTERED
(
      [reservaton_details_id] ASC
)
)
GO

ALTER TABLE [dbo].[CompanyReservationDetails]  WITH CHECK ADD
CONSTRAINT [FK_CompanyReservationDetails_CompanyReservations] FOREIGN
KEY([reservation_id])
REFERENCES [dbo].[CompanyReservations] ([company_reservation_id])
GO
ALTER TABLE [dbo].[CompanyReservationDetails] CHECK CONSTRAINT
[FK_CompanyReservationDetails_CompanyReservations]
GO
ALTER TABLE [dbo].[CompanyReservationDetails]  WITH CHECK ADD
CONSTRAINT [FK_CompanyReservationDetails_CompanyReservationTables]
FOREIGN KEY([reservaton_details_id])
REFERENCES [dbo].[CompanyReservationTables] ([reservation_details_id])
GO
ALTER TABLE [dbo].[CompanyReservationDetails] CHECK CONSTRAINT
[FK_CompanyReservationDetails_CompanyReservationTables]
GO

ALTER TABLE [dbo].[CompanyReservationDetails]  WITH CHECK ADD CONSTRAINT
[CK_positive] CHECK  ([people_count]>(1))
GO
ALTER TABLE [dbo].[CompanyReservationDetails] CHECK CONSTRAINT
```

```
[CK_positive]
GO
```

CompanyReservationEmployees

Opisuje którzy pracownicy są zawarcji w jakich rezerwacjach.

- **PK** reservation_details_id **int** - ID szczegółów rezerwacji, do których przypisany ma być dany pracownik
- **PK** employee_customer_id **int** - ID pracownika jako klienta prywatnego, który ma być zawarty w rezerwacji

```
CREATE TABLE [dbo].[CompanyReservationEmployees](
      [reservation_details_id] [int] NOT NULL,
      [employee_customer_id] [int] NOT NULL,
 CONSTRAINT [PK_CompanyReservationEmployees] PRIMARY KEY CLUSTERED
(
      [reservation_details_id] ASC,
      [employee_customer_id] ASC
)
)
GO


ALTER TABLE [dbo].[CompanyReservationEmployees]  WITH CHECK ADD
CONSTRAINT [FK_CompanyReservationEmployees_CompanyReservationDetails]
FOREIGN KEY([named_reservation_details_id])
REFERENCES [dbo].[CompanyReservationDetails] ([reservaton_details_id])
GO
ALTER TABLE [dbo].[CompanyReservationEmployees] CHECK CONSTRAINT
[FK_CompanyReservationEmployees_CompanyReservationDetails]
GO
ALTER TABLE [dbo].[CompanyReservationEmployees]  WITH CHECK ADD
CONSTRAINT [FK_CompanyReservationEmployees_PrivateCustomers] FOREIGN
KEY([employee_customer_id])
REFERENCES [dbo].[PrivateCustomers] ([customer_id])
GO
ALTER TABLE [dbo].[CompanyReservationEmployees] CHECK CONSTRAINT
[FK_CompanyReservationEmployees_PrivateCustomers]
GO
```

CompanyReservations

Opisuje rezerwacje firmowe.

- **PK** company_reservation_id **int** - ID rezerwacji firmowej
- **FK** customer_id **int** - ID klienta (firmy)
- start_time **datetime** - czas początku rezerwacji
- end_time **datetime** - czas końca rezerwacji

Warunki integralnościowe:

- end_time musi być większe od start_time:
  `CONSTRAINT [CK_chronological_date] CHECK  ([end_time]>[start_time])`

```
CREATE TABLE [dbo].[CompanyReservations](
	[company_reservation_id] [int] IDENTITY(1,1) NOT NULL,
	[customer_id] [int] NOT NULL,
	[start_time] [datetime] NOT NULL,
	[end_time] [datetime] NOT NULL,
 CONSTRAINT [PK_CompanyReservations] PRIMARY KEY CLUSTERED
(
	[company_reservation_id] ASC
)
)
GO

ALTER TABLE [dbo].[CompanyReservations]  WITH CHECK ADD  CONSTRAINT
[FK_CompanyReservations_CompanyCustomers] FOREIGN KEY([customer_id])
REFERENCES [dbo].[CompanyCustomers] ([customer_id])
GO
ALTER TABLE [dbo].[CompanyReservations] CHECK CONSTRAINT
[FK_CompanyReservations_CompanyCustomers]
GO

--DATE CHRONOLOGICAL
ALTER TABLE [dbo].[CompanyReservations]  WITH CHECK ADD  CONSTRAINT
[CK_chronological_date] CHECK  ([end_time]>[start_time])
GO
ALTER TABLE [dbo].[CompanyReservations] CHECK CONSTRAINT
[CK_chronological_date]
GO
```

CompanyReservationTables

Przypisanie stolików do poszczególnych części rezerwacji firmowej.

- **PK** reservation_details_id **int** - ID szczegółów rezerwacji
- **FK** table_id **int** - ID stolika przypisanego do danej części rezerwacji

```
CREATE TABLE [dbo].[CompanyReservationTables](
	[reservation_details_id] [int] NOT NULL,
	[table_id] [int] NOT NULL,
 CONSTRAINT [PK_CompanyReservationTables] PRIMARY KEY CLUSTERED
(
	[reservation_details_id] ASC
)
)
GO

ALTER TABLE [dbo].[CompanyReservationTables]  WITH CHECK ADD  CONSTRAINT
[FK_CompanyReservationTables_Tables] FOREIGN KEY([table_id])
REFERENCES [dbo].[Tables] ([table_id])
GO
ALTER TABLE [dbo].[CompanyReservationTables] CHECK CONSTRAINT
[FK_CompanyReservationTables_Tables]
GO
```

Countries

Słownik dla krajów.

- **PK** country_id **int** - ID kraju
- name **nvarchar(50)** - nazwa kraju

Warunki integralnościowe:

- name jest unikalne
  `CONSTRAINT [unique_name] UNIQUE NONCLUSTERED`

```
CREATE TABLE [dbo].[Countries](
	[country_id] [int] IDENTITY(1,1) NOT NULL,
	[name] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_Countries] PRIMARY KEY CLUSTERED
(
	[country_id] ASC
),
 CONSTRAINT [unique_name] UNIQUE NONCLUSTERED
(
	[name] ASC
))
GO
```

Customers

Generalizacja klientów - wspólne informacje o klientach prywatnych i firmowych.

- **PK** customer_id **int** - ID klienta
- phone **varchar(15)** - nr telefonu
- street **nvarchar(50)** - ulica klienta
- city_id **int** - miasto klienta

Warunki integralnościowe:

- phone musi składać się z samych cyfr:
  `CONSTRAINT [CK_phone] CHECK  ([phone] not like '%[^0-9]%')`

```
CREATE TABLE [dbo].[Customers](
	[customer_id] [int] IDENTITY(1,1) NOT NULL,
	[phone] [varchar](15) NOT NULL,
	[street] [nvarchar](50) NOT NULL,
	[city_id] [int] NOT NULL,
 CONSTRAINT [PK_Customers] PRIMARY KEY CLUSTERED
(
	[customer_id] ASC
)
)
GO

ALTER TABLE [dbo].[Customers]  WITH CHECK ADD  CONSTRAINT
[FK_Customers_Cities] FOREIGN KEY([city_id])
REFERENCES [dbo].[Cities] ([city_id])
GO
ALTER TABLE [dbo].[Customers] CHECK CONSTRAINT [FK_Customers_Cities]
GO

--PHONE
ALTER TABLE [dbo].[Customers]  WITH CHECK ADD  CONSTRAINT [CK_phone]
CHECK  ([phone] not like '%[^0-9]%')
GO
ALTER TABLE [dbo].[Customers] CHECK CONSTRAINT [CK_phone]
GO
```

DiscountParamHistory

Opisuje zmiany obowiązujących wartości parametrów w czasie.

- **PK** discount_id **int** - ID zniżki, której dotyczy
- param_id **nchar(10)** - ID parametru
- value **int** - wartość parametru
- active_from **datetime** - od kiedy ważny jest parametr
- active_to **datetime** NULL- do kiedy ważny jest parametr

Warunki integralnościowe:

- active_to musi być większe od active_from:
  `CONSTRAINT [CK_chronological_date] CHECK ([active_to]>[active_from])`
- active from domyślnie jest dzisiejszą datą `DEFAULT GETDATE(),`

```
CREATE TABLE [dbo].[DiscountParamHistory](
	[discount_id] [int] NOT NULL,
	[param_id] [nchar](10) NOT NULL,
	[value] [int] NOT NULL,
	[active_from] [datetime] NOT NULL DEFAULT GETDATE(),
	[active_to] [datetime] NULL,
 CONSTRAINT [PK_DiscountParamHistory] PRIMARY KEY CLUSTERED
(
	[discount_id] ASC
)
)
GO

ALTER TABLE [dbo].[DiscountParamHistory]  WITH CHECK ADD  CONSTRAINT
[FK_DiscountParamHistory_DiscountParams] FOREIGN KEY([param_id])
REFERENCES [dbo].[DiscountParams] ([param_id])
GO
ALTER TABLE [dbo].[DiscountParamHistory] CHECK CONSTRAINT
[FK_DiscountParamHistory_DiscountParams]
GO

--DATE CHRONOLOGICAL
ALTER TABLE [dbo].[DiscountParamHistory]  WITH CHECK ADD  CONSTRAINT
[CK_chronological_date] CHECK  ([active_to]>[active_from])
GO
ALTER TABLE [dbo].[DiscountParamHistory] CHECK CONSTRAINT
[CK_chronological_date]
GO
```

DiscountParams

Słownik parametrów.

- **PK** param_id **nchar(10)** - ID parametru
- value **int** - wartość parametru

Warunki integralnościowe:

- value musi być większe od 0:
  `CONSTRAINT [CK_positive] CHECK  ([value]>(0))`

```
CREATE TABLE [dbo].[DiscountParams](
      [param_id] [varchar](10) NOT NULL,
      [value] [int] NOT NULL,
 CONSTRAINT [PK_DiscountParams] PRIMARY KEY CLUSTERED
(
      [param_id] ASC
)
)
GO

ALTER TABLE [dbo].[DiscountParamHistory]  WITH CHECK ADD CONSTRAINT
[CK_positive] CHECK  ([value]>(0))
GO
ALTER TABLE [dbo].[DiscountParamHistory] CHECK CONSTRAINT [CK_positive]
GO
```

Dishes

Wszystkie dania.

- **PK** dish_id **int** - ID dania
- name **nvarchar(50)** - nazwa dania
- **FK** category_id **int** - ID kategorii
- description **text** NULL - opis dania
- seafood **bit** - czy to owoce morza, które podlega szczególnym zasadom zamawiania

Warunki integralnościowe:

- name jest unikalne
  `CONSTRAINT [unique_name] UNIQUE NONCLUSTERED`

```
CREATE TABLE [dbo].[Dishes](
	[dish_id] [int] IDENTITY(1,1) NOT NULL,
	[name] [nvarchar](50) NOT NULL,
	[category_id] [int] NOT NULL,
	[description] [text] NULL,
	[seafood] [bit] NOT NULL,
 CONSTRAINT [PK_Dishes] PRIMARY KEY CLUSTERED
(
	[dish_id] ASC
),
CONSTRAINT [unique_name] UNIQUE NONCLUSTERED
(
	[name] ASC
)
)
GO

ALTER TABLE [dbo].[Dishes]  WITH CHECK ADD  CONSTRAINT
[FK_Dishes_Categories] FOREIGN KEY([category_id])
REFERENCES [dbo].[Categories] ([category_id])
GO
ALTER TABLE [dbo].[Dishes] CHECK CONSTRAINT [FK_Dishes_Categories]
GO
```

Employees

Dane pracowników.

- **PK** employee_id **int** - ID pracownika
- **FK** reports_to **int** - ID przełożonego
- firstname **nvarchar(50)** - imię
- lastname **nvarchar(50)** - nazwisko
- email **varchar(50)** - adres email

Warunki integralnościowe:

- email musi być postaci sth@example.ex:
  `CONSTRAINT [CK_email] CHECK  ([email] like '%@%[.]%')`

```
CREATE TABLE [dbo].[Employees](
	[employee_id] [int] IDENTITY(1,1) NOT NULL,
	[reports_to] [int] NULL,
	[firstname] [nvarchar](50) NOT NULL,
	[lastname] [nvarchar](50) NOT NULL,
	[email] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Employees] PRIMARY KEY CLUSTERED
(
	[employee_id] ASC
),
CONSTRAINT [unique_email] UNIQUE NONCLUSTERED
(
	[email] ASC
)
)
GO

ALTER TABLE [dbo].[Employees]  WITH CHECK ADD  CONSTRAINT
[FK_Employees_Employees] FOREIGN KEY([reports_to])
REFERENCES [dbo].[Employees] ([employee_id])
GO
ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [FK_Employees_Employees]
GO

--EMAIL
ALTER TABLE [dbo].[Employees]  WITH CHECK ADD  CONSTRAINT [CK_email]
CHECK  ([email] like '%@%[.]%')
GO
ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [CK_email]
GO
```

OrderDetails

Szczegóły zamówień.

- **PK** order_id **int** - ID zamówienia
- **PK** planned_dish_id **int** - ID zamawianej pozycji
- quantity **int** - ile razy dano danie w zamówieniu

Warunki integralnościowe:

- quantity musi być większe od 0:
  `CONSTRAINT [CK_positive] CHECK  ([quantity]>(0))`

```
CREATE TABLE [dbo].[OrderDetails](
	[order_id] [int] NOT NULL,
	[planned_dish_id] [int] NOT NULL,
	[quantity] [int] NOT NULL,
 CONSTRAINT [PK_OrderDetails] PRIMARY KEY CLUSTERED
(
	[order_id] ASC,
	[planned_dish_id] ASC
)
)
GO

ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT
[FK_OrderDetails_Orders] FOREIGN KEY([order_id])
REFERENCES [dbo].[Orders] ([order_id])
GO
ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT
[FK_OrderDetails_Orders]
GO
ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT
[FK_OrderDetails_PlannedDishes] FOREIGN KEY([planned_dish_id])
REFERENCES [dbo].[PlannedDishes] ([planned_dish_id])
GO
ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT
[FK_OrderDetails_PlannedDishes]
GO

ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT
[CK_positive] CHECK  ([quantity]>(0))
GO
ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT [CK_positive]
GO
```

Orders

Zamówienia.

- **PK** order_id **int** - ID zamówienia
- **FK** customer_id **int** - ID klienta, który zamawia
- **FK** employee_id **int** NULL - ID pracownika, który obsługuje zamówienie
- order_date **datetime** - data zamówienia
- target_date **datetime** - na kiedy zamówione
- paid_in_advance **bit** - czy zamówienie opłacone z góry

Warunki integralnościowe:

- target_date musi być większe od order_date: `CONSTRAINT [CK_chronological_date] CHECK ([target_date]>[order_date])`
- order_date domyślnie jest teraźniejszą datą `DEFAULT GETDATE()`

```
CREATE TABLE [dbo].[Orders](
	[order_id] [int] IDENTITY(1,1) NOT NULL,
	[customer_id] [int] NOT NULL,
	[employee_id] [int] NULL,
	[order_date] [datetime] NOT NULL DEFAULT GETDATE(),
	[target_date] [datetime] NOT NULL,
	[paid_in_advance] [bit] NOT NULL,
 CONSTRAINT [PK_Orders] PRIMARY KEY CLUSTERED
(
	[order_id] ASC
)
)
GO

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT
[FK_Orders_Customers] FOREIGN KEY([customer_id])
REFERENCES [dbo].[Customers] ([customer_id])
GO
ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [FK_Orders_Customers]
GO
ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT
[FK_Orders_Employees] FOREIGN KEY([employee_id])
REFERENCES [dbo].[Employees] ([employee_id])
GO
ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [FK_Orders_Employees]
GO
ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT
[FK_Orders_PrivateReservations] FOREIGN KEY([reservation_id])
REFERENCES [dbo].[PrivateReservations] ([reservation_id])
```

```
GO
ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT
[FK_Orders_PrivateReservations]
GO

--DATE CHRONOLOGICAL
ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT
[CK_chronological_date] CHECK  ([target_date]>[order_date])
GO
ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [CK_chronological_date]
GO
```

OrderStatuses

Słownik statusów zamówień.

- **PK** order_status_id **int** - ID statusu
- order_status **nvarchar(50)** - nazwa statusu

Warunki integralnościowe:

- order_status jest unikalne
  `CONSTRAINT [order_status] UNIQUE NONCLUSTERED`

```
CREATE TABLE [dbo].[OrderStatuses](
	[order_status_id] [int] IDENTITY(1,1) NOT NULL,
	[order_status] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_OrderStatuses] PRIMARY KEY CLUSTERED
(
	[order_status_id] ASC
),
CONSTRAINT [unique_status] UNIQUE NONCLUSTERED
(
	[order_status] ASC
)
)
GO
```

OrderStatusHistory

Historia statusów zamówień.

- **PK** order_id **int** - ID zamówienia
- **PK** order_status_id **int** - ID statusu
- datetime **datetime** - czas zarejestrowania statusu

Warunki integralnościowe:

- datetime domyślnie jest dzisiejszą datą `DEFAULT GETDATE()`

```
CREATE TABLE [dbo].[OrderStatusHistory](
	[order_id] [int] NOT NULL,
	[order_status_id] [int] NOT NULL,
	[datetime] [datetime] NOT NULL DEFAULT GETDATE(),
 CONSTRAINT [PK_OrderStatusHistory] PRIMARY KEY CLUSTERED
(
	[order_id] ASC,
	[order_status_id] ASC
))
GO

ALTER TABLE [dbo].[OrderStatusHistory]  WITH CHECK ADD  CONSTRAINT
[FK_OrderStatusHistory_Orders] FOREIGN KEY([order_id])
REFERENCES [dbo].[Orders] ([order_id])
GO
ALTER TABLE [dbo].[OrderStatusHistory] CHECK CONSTRAINT
[FK_OrderStatusHistory_Orders]
GO
ALTER TABLE [dbo].[OrderStatusHistory]  WITH CHECK ADD  CONSTRAINT
[FK_OrderStatusHistory_OrderStatuses] FOREIGN KEY([order_status_id])
REFERENCES [dbo].[OrderStatuses] ([order_status_id])
GO
ALTER TABLE [dbo].[OrderStatusHistory] CHECK CONSTRAINT
[FK_OrderStatusHistory_OrderStatuses]
GO
```

PermanentDiscounts

Tabela zniżek przyznawanych na zawsze.

- **PK** customer_id **int** - ID klienta
- **PK** discount_id **int** - ID zniżki
- discount **real** - wartość przyznanej zniżki

Warunki integralnościowe:

- discount musi być pomiędzy 0 a 1:
  `CONSTRAINT [CK_reasonable_discount] CHECK (discount between 0 and 1 )`

```
CREATE TABLE [dbo].[PermanentDiscounts](
      [customer_id] [int] NOT NULL,
      [discount_id] [int] NOT NULL,
      [discount] [real] NOT NULL,
 CONSTRAINT [PK_PermanentDiscounts] PRIMARY KEY CLUSTERED
(
      [customer_id] ASC
))
GO

ALTER TABLE [dbo].[PermanentDiscounts]  WITH CHECK ADD  CONSTRAINT
[FK_PermanentDiscounts_DiscountParamHistory] FOREIGN KEY([discount_id])
REFERENCES [dbo].[DiscountParamHistory] ([discount_id])
GO
ALTER TABLE [dbo].[PermanentDiscounts] CHECK CONSTRAINT
[FK_PermanentDiscounts_DiscountParamHistory]
GO

--REASONABLE DISCOUNT
ALTER TABLE [dbo].[PermanentDiscounts]  WITH CHECK ADD  CONSTRAINT
[CK_reasonable_discount] CHECK  (discount between 0 and 1 )
GO
ALTER TABLE [dbo].[PermanentDiscounts] CHECK CONSTRAINT
[CK_reasonable_discount]
GO
```

PlannedDishes

Rozplanowanie kiedy, jakie danie dostępne jest po jakiej cenie. Oferty dań.

- **PK** planned_dish_id **int** - ID planowanego dania
- **FK** dish_id **int** - ID dania
- price **money** - cena
- active_from **datetime** - od kiedy obowiązuje oferta
- active_to **datetime** NULL- do kiedy obowiązuje oferta

Warunki integralnościowe:

- price musi być większe od 0:
  `CONSTRAINT [CK_positive] CHECK  ([price]>0)`
- active_to musi być większe od active_from:
  `CONSTRAINT [CK_chronological_date] CHECK ([active_to]>[active_from])`
- active_from domyślnie jest teraźniejszą datą `DEFAULT GETDATE()`

```
CREATE TABLE [dbo].[PlannedDishes](
      [planned_dish_id] [int] IDENTITY(1,1) NOT NULL,
      [dish_id] [int] NOT NULL,
      [price] [money] NOT NULL,
      [active_from] [datetime] NOT NULL DEFAULT GETDATE(),
      [active_to] [datetime] NULL,
 CONSTRAINT [PK_PlannedDishes] PRIMARY KEY CLUSTERED
(
      [planned_dish_id] ASC
))
GO

ALTER TABLE [dbo].[PlannedDishes]  WITH CHECK ADD  CONSTRAINT
[FK_PlannedDishes_Dishes] FOREIGN KEY([dish_id])
REFERENCES [dbo].[Dishes] ([dish_id])
GO
ALTER TABLE [dbo].[PlannedDishes] CHECK CONSTRAINT
[FK_PlannedDishes_Dishes]
GO

--DATE CHRONOLOGICAL
ALTER TABLE [dbo].[PlannedDishes]  WITH CHECK ADD  CONSTRAINT
[CK_chronological_date] CHECK  (([active_to]>[active_from]) or
[active_to] is null)
GO
ALTER TABLE [dbo].[PlannedDishes] CHECK CONSTRAINT
[CK_chronological_date]
GO
```

```
--POSITIVE VALUE
ALTER TABLE [dbo].[PlannedDishes]  WITH CHECK ADD  CONSTRAINT
[CK_positive] CHECK  ([price]>0)
GO
ALTER TABLE [dbo].[PlannedDishes] CHECK CONSTRAINT [CK_positive]
GO
```

PrivateCustomers

Tabela z danymi klientów indywidualnych.

- **PK** customer_id **int** - ID klienta
- firstname **nvarchar(50)** - imię
- lastname **nvarchar(50)** - nazwisko
- email **varchar(50)** - adres email

Warunki integralnościowe:

- email musi być postaci sth@example.ex:
  `CONSTRAINT [CK_email] CHECK  ([email] like '%@%[.]%')`

```
CREATE TABLE [dbo].[PrivateCustomers](
	[customer_id] [int] NOT NULL,
	[firstname] [nvarchar](50) NOT NULL,
	[lastname] [nvarchar](50) NOT NULL,
	[email] [varchar](50) NOT NULL,
 CONSTRAINT [PK_PrivateCustomers] PRIMARY KEY CLUSTERED
(
	[customer_id] ASC
),
CONSTRAINT [unique_email] UNIQUE NONCLUSTERED
(
	[email] ASC
)
)
GO

ALTER TABLE [dbo].[PrivateCustomers]  WITH CHECK ADD  CONSTRAINT
[FK_PrivateCustomers_Customers] FOREIGN KEY([customer_id])
REFERENCES [dbo].[Customers] ([customer_id])
GO
ALTER TABLE [dbo].[PrivateCustomers] CHECK CONSTRAINT
[FK_PrivateCustomers_Customers]
GO
ALTER TABLE [dbo].[PrivateCustomers]  WITH CHECK ADD  CONSTRAINT
[FK_PrivateCustomers_PermanentDiscounts] FOREIGN KEY([customer_id])
REFERENCES [dbo].[PermanentDiscounts] ([customer_id])
GO
ALTER TABLE [dbo].[PrivateCustomers] CHECK CONSTRAINT
[FK_PrivateCustomers_PermanentDiscounts]
GO

--EMAIL
ALTER TABLE [dbo].[PrivateCustomers]  WITH CHECK ADD  CONSTRAINT
```

```
[CK_email] CHECK  ([email] like '%@%[.]%')
GO
ALTER TABLE [dbo].[PrivateCustomers] CHECK CONSTRAINT [CK_email]
GO
```

PrivateReservations

Rezerwacje indywidualnych klientów

- **PK** private_reservation_id **int** - ID rezerwacji indywidualnej
- **FK** customer_id **int** - ID klienta
- **FK** customer_id **int** - ID zamówienia
- people_count **int** - ile osób w rezerwacji
- start_time **datetime** - czas początku rezerwacji
- end_time **datetime** - czas końca rezerwacji
- table_id **int** NULL - ID stolika

Warunki integralnościowe:

- people_count musi być większe od 1:
  `CONSTRAINT [CK_positive] CHECK  ([people_count]>1)`
- end_time musi być większe od start_time:
  `CONSTRAINT [CK_chronological_date] CHECK ([end_time]>[start_time])`

```
CREATE TABLE [dbo].[PrivateReservations](
	[private_reservation_id] [int] NOT NULL,
	[customer_id] [int] NOT NULL,
	[order_id] [int] NOT NULL,
	[people_count] [int] NOT NULL,
	[start_time] [datetime] NOT NULL,
	[end_time] [datetime] NOT NULL,
	[table_id] [int] NULL,
 CONSTRAINT [PK_PrivateReservations] PRIMARY KEY CLUSTERED
(
	[private_reservation_id] ASC
)
)
GO

ALTER TABLE [dbo].[PrivateReservations]  WITH CHECK ADD  CONSTRAINT
[FK_PrivateReservations_Orders] FOREIGN KEY([order_id])
REFERENCES [dbo].[Orders] ([order_id])
GO
ALTER TABLE [dbo].[PrivateReservations] CHECK CONSTRAINT
[FK_PrivateReservations_Orders]
GO
ALTER TABLE [dbo].[PrivateReservations]  WITH CHECK ADD  CONSTRAINT
[FK_PrivateReservations_PrivateCustomers] FOREIGN KEY([customer_id])
REFERENCES [dbo].[PrivateCustomers] ([customer_id])
GO
ALTER TABLE [dbo].[PrivateReservations] CHECK CONSTRAINT
[FK_PrivateReservations_PrivateCustomers]
```

```
GO
ALTER TABLE [dbo].[PrivateReservations]  WITH CHECK ADD  CONSTRAINT
[FK_PrivateReservations_Tables] FOREIGN KEY([table_id])
REFERENCES [dbo].[Tables] ([table_id])
GO
ALTER TABLE [dbo].[PrivateReservations] CHECK CONSTRAINT
[FK_PrivateReservations_Tables]
GO

--DATE CHRONOLOGICAL
ALTER TABLE [dbo].[PrivateReservations]  WITH CHECK ADD  CONSTRAINT
[CK_chronological_date] CHECK  ([end_time]>[start_time])
GO
ALTER TABLE [dbo].[PrivateReservations] CHECK CONSTRAINT
[CK_chronological_date]
GO

--POSITIVE VALUE
ALTER TABLE [dbo].[PrivateReservations]  WITH CHECK ADD  CONSTRAINT
[CK_positive] CHECK  ([people_count]>1)
GO
ALTER TABLE [dbo].[PrivateReservations] CHECK CONSTRAINT [CK_positive]
GO
```

ReservationsParams

Słownik, zawiera wartości, które używane są przy ustalaniu czy klient może złożyć rezerwację.

- **PK** param_id **int** - ID parametru
- name **varchar(20)** - nazwa parametru
- value **int** - wartość parametru

Warunki integralnościowe:

- name jest unikalne
  `CONSTRAINT [name] UNIQUE NONCLUSTERED`
- value musi być większe od 0:
  `CONSTRAINT [CK_positive] CHECK  ([value]>0)`

```
CREATE TABLE [dbo].[ReservationParams](
	[param_id] [int] NOT NULL,
	[name] [varchar](20) NOT NULL,
	[value] [int] NOT NULL,
 CONSTRAINT [PK_ReservationParams] PRIMARY KEY CLUSTERED
(
	[param_id] ASC
)
CONSTRAINT [unique_status] UNIQUE NONCLUSTERED
(
	[name] ASC
)
)
GO

--POSITIVE VALUE
ALTER TABLE [dbo].[ReservationParams]  WITH CHECK ADD  CONSTRAINT
[CK_positive] CHECK  ([value]>0)
GO
ALTER TABLE [dbo].[ReservationParams] CHECK CONSTRAINT [CK_positive]
GO
```

ReservationsStatuses

Słownik statusów rezerwacji

- **PK** reservation_status_id **int** - ID statusu
- status **nvarchar(50)** - nazwa statusu

Warunki integralnościowe:

- status jest unikalne
  `CONSTRAINT [unique_status] UNIQUE NONCLUSTERED`

```
CREATE TABLE [dbo].[ReservationsStatuses](
	[reservation_status_id] [int] NOT NULL,
	[status] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_ReservationsStatuses] PRIMARY KEY CLUSTERED
(
	[reservation_status_id] ASC
),
CONSTRAINT [unique_status] UNIQUE NONCLUSTERED
(
	[status] ASC
)
)
GO
```

ReservationStatusHistory

Historia statusów rezerwacji.

- **PK** reservation_status_id **int** - ID statusu
- **PK** reservation_id **int** - ID rezerwacji
- datetime **datetime** - czas zarejestrowania statusu

Warunki integralnościowe:

- datetime domyślnie jest dzisiejszą datą `DEFAULT GETDATE(),`

```
CREATE TABLE [dbo].[ReservationStatusHistory](
	[reservation_status_id] [int] NOT NULL,
	[reservation_id] [int] NOT NULL,
	[datetime] [datetime] NOT NULL DEFAULT GETDATE(),
 CONSTRAINT [PK_ReservationStatusHistory] PRIMARY KEY CLUSTERED
(
	[reservation_status_id] ASC,
	[reservation_id] ASC
)
)
GO

ALTER TABLE [dbo].[ReservationStatusHistory]  WITH CHECK ADD  CONSTRAINT
[FK_ReservationStatusHistory_CompanyReservations1] FOREIGN
KEY([reservation_id])
REFERENCES [dbo].[CompanyReservations] ([company_reservation_id])
GO
ALTER TABLE [dbo].[ReservationStatusHistory] CHECK CONSTRAINT
[FK_ReservationStatusHistory_CompanyReservations1]
GO
ALTER TABLE [dbo].[ReservationStatusHistory]  WITH CHECK ADD  CONSTRAINT
[FK_ReservationStatusHistory_PrivateReservations] FOREIGN
KEY([reservation_id])
REFERENCES [dbo].[PrivateReservations] ([private_reservation_id])
GO
ALTER TABLE [dbo].[ReservationStatusHistory] CHECK CONSTRAINT
[FK_ReservationStatusHistory_PrivateReservations]
GO
ALTER TABLE [dbo].[ReservationStatusHistory]  WITH CHECK ADD  CONSTRAINT
[FK_ReservationStatusHistory_ReservationsStatuses] FOREIGN
KEY([reservation_status_id])
REFERENCES [dbo].[ReservationsStatuses] ([reservation_status_id])
GO
ALTER TABLE [dbo].[ReservationStatusHistory] CHECK CONSTRAINT
[FK_ReservationStatusHistory_ReservationsStatuses]
```

GO

Tables

Stoliki.

- **PK** table_id **int** - ID statusu
- seats **int** - liczba miejsc przy stoliku
- is_active **bit** - czy stolik jest aktywny (czy może zostać przydzielony klientowi)

Warunki integralnościowe:

- seats musi być większe od 0:
  `CONSTRAINT [CK_positive] CHECK  ([seats]>0)`

```
CREATE TABLE [dbo].[Tables](
	[table_id] [int] NOT NULL,
	[seats] [int] NOT NULL,
	[is_active] [bit] NOT NULL,
 CONSTRAINT [PK_Tables] PRIMARY KEY CLUSTERED
(
	[table_id] ASC
)
)
GO

--POSITIVE VALUE
ALTER TABLE [dbo].[Tables]  WITH CHECK ADD  CONSTRAINT [CK_positive]
CHECK  ([seats]>(0))
GO
ALTER TABLE [dbo].[Tables] CHECK CONSTRAINT [CK_positive]
GO
```

TemporaryDiscounts

Tabela zniżek przyznawanych na pewien okres.

- **PK** customer_id **int** - ID klienta
- **PK** discount_id **int** - ID zniżki
- discount **real** - wartość przyznanej zniżki
- active_from **datetime** - od kiedy obowiązuje danego klienta zniżka
- active_to **datetime** - do kiedy obowiązuje klienta zniżka

Warunki integralnościowe:

- discount musi być pomiędzy 0 a 1:
  `CONSTRAINT [CK_reasonable_discount] CHECK (discount between 0 and 1 )`
- active_to musi być większe od active_from:
  `CONSTRAINT [CK_chronological_date] CHECK ([active_to]>[active_from])`

```
CREATE TABLE [dbo].[TemporaryDiscounts](
	[customer_id] [int] NOT NULL,
	[discount_id] [int] NOT NULL,
	[discount] [real] NOT NULL,
	[active_from] [datetime] NOT NULL,
	[active_to] [datetime] NOT NULL,
 CONSTRAINT [PK_TemporaryDiscounts] PRIMARY KEY CLUSTERED
(
	[customer_id] ASC,
	[discount_id] ASC
)
)
GO

ALTER TABLE [dbo].[TemporaryDiscounts]  WITH CHECK ADD  CONSTRAINT
[FK_TemporaryDiscounts_DiscountParamHistory] FOREIGN KEY([discount_id])
REFERENCES [dbo].[DiscountParamHistory] ([discount_id])
GO
ALTER TABLE [dbo].[TemporaryDiscounts] CHECK CONSTRAINT
[FK_TemporaryDiscounts_DiscountParamHistory]
GO
ALTER TABLE [dbo].[TemporaryDiscounts]  WITH CHECK ADD  CONSTRAINT
[FK_TemporaryDiscounts_PrivateCustomers] FOREIGN KEY([customer_id])
REFERENCES [dbo].[PrivateCustomers] ([customer_id])
GO
ALTER TABLE [dbo].[TemporaryDiscounts] CHECK CONSTRAINT
[FK_TemporaryDiscounts_PrivateCustomers]
GO
```

```
--REASONABLE DISCOUNT
ALTER TABLE [dbo].[TemporaryDiscounts]  WITH CHECK ADD  CONSTRAINT
[CK_reasonable_discount] CHECK  (discount between 0 and 1 )
GO
ALTER TABLE [dbo].[TemporaryDiscounts] CHECK CONSTRAINT
[CK_reasonable_discount]
GO

--DATE CHRONOLOGICAL
ALTER TABLE [dbo].[TemporaryDiscounts]  WITH CHECK ADD  CONSTRAINT
[CK_chronological_date] CHECK  ([active_to]>[active_from])
GO
ALTER TABLE [dbo].[TemporaryDiscounts] CHECK CONSTRAINT
[CK_chronological_date]
GO
```

Widoki

AllEmployeeView - dane wszystkich pracowników

```
CREATE VIEW [dbo].[AllEmployeesView]
AS
SELECT dbo.Employees.firstname, dbo.Employees.lastname, dbo.Employees.phone,
dbo.Employees.email, Employeer.employee_id
FROM  dbo.Employees INNER JOIN
dbo.Employees Employeer ON dbo.Employees.reports_to = Employeer.employee_id
GO
```

AllCustomersView - wszyscy klienci

```
CREATE view [dbo].[AllCustomersView] as
SELECT
	c.customer_id,
	phone,
	street,
	Cities.name [City],
	firstname + ' ' + lastname as [Name],
	'private' as [Customer type]
FROM
	Customers c
	inner join Cities on c.city_id = Cities.city_id
	inner join PrivateCustomers pc on c.customer_id=pc.customer_id
UNION
SELECT
	c.customer_id,
	phone,
	street,
	Cities.name as [City],
	cc.name as [Name],
	'company' as [Customer type]
FROM
	Customers c
	inner join Cities on c.city_id = Cities.city_id
	inner join CompanyCustomers cc on c.customer_id=cc.customer_id
GO
```

*Monthly*AmountOfCompanyCustomerOrdersView - *liczba wszystkich zamówień firmowych zgrupowane miesięcznie*

```
CREATE VIEW [dbo].[Monthly]AmountOfCompanyCustomerOrdersView]
AS
SELECT Year, Month, SUM(Amount) AS Amount
FROM     dbo.AmountOfCompanyCustomerOrdersView
GROUP BY Year, Month
GO
```

AmountOfCompanyCustomerOrdersView - *liczba wszystkich zamówień firmowych pogrupowanych po godzinach*

```
CREATE VIEW [dbo].[AmountOfCompanyCustomerOrdersView]
AS
SELECT COUNT(*) AS Amount, YEAR(dbo.Orders.order_date) AS Year,
MONTH(dbo.Orders.order_date) AS Month, DATEPART(dw, dbo.Orders.order_date) AS
Day, DATEPART(hh, dbo.Orders.order_date) AS Hour
FROM     dbo.Orders INNER JOIN
              dbo.Customers ON dbo.Customers.customer_id = dbo.Orders.customer_id
INNER JOIN
              dbo.CompanyCustomers ON dbo.CompanyCustomers.customer_id =
dbo.Customers.customer_id
GROUP BY YEAR(dbo.Orders.order_date), MONTH(dbo.Orders.order_date),
DATEPART(dw, dbo.Orders.order_date), DATEPART(hh, dbo.Orders.order_date)
GO
```

MonthlyCompanyCustomerOrderDetailsReport - szczegółowy raport miesięczny o zamówieniach klientów firmowych zgrupowane po zamówieniach

```
CREATE VIEW [MonthlyCompanyCustomerOrderDetailsReport] AS
SELECT O.order_id, O.customer_id, CC.name as 'customerName', D.name,
PD.price * OD.quantity as dishQuantityCost, PD.price, OD.quantity,
target_date, CONVERT(TIME,target_date) as 'Time', year(target_date)
as 'year', month(target_date) as 'month'
FROM Orders O
INNER JOIN CompanyCustomers CC on O.customer_id = CC.customer_id
INNER JOIN OrderDetails OD on O.order_id = OD.order_id
```

```
INNER JOIN PlannedDishes PD on OD.planned_dish_id =
PD.planned_dish_id
```

WeeklyCompanyCustomerOrderDetailsReport - szczegółowy raport miesięczny o zamówieniach klientów firmowych zgrupowane po zamówieniach

```
CREATE VIEW [WeeklyCompanyCustomerOrderDetailsReport] AS
SELECT O.order_id, O.customer_id, CC.name as 'customerName', D.name,
PD.price * OD.quantity as dishQuantityCost, PD.price, OD.quantity,
target_date, CONVERT(TIME,target_date) as 'Time', year(target_date)
as 'year', datepart(ww, target_date) as 'week'
FROM Orders O
INNER JOIN CompanyCustomers CC on O.customer_id = CC.customer_id
INNER JOIN OrderDetails OD on O.order_id = OD.order_id
INNER JOIN PlannedDishes PD on OD.planned_dish_id =
PD.planned_dish_id
Inner join Dishes D on D.dish_id= PD.dish_id
```

*Monthly*CompanyCustomerOrdersStatisticsView - *statystyka miesięczna sumy*

wydanej przez firmy oraz ilosc złożonych zamówień zgrupowanych po klientach firmowych

```
CREATE VIEW [Monthly]CompanyCustomerOrdersStatisticsView] AS
SELECT O.customer_id , C.name,
SUM(PD.price * OD.quantity) AS 'totalPricePaid',
year(target_date) as 'year', month(target_date) as 'month'
FROM Orders O
JOIN Companies C on O.customer_id = C.customer_id
INNER JOIN OrderDetails OD on O.order_id = OD.order_id
INNER JOIN PlannedDishes PD on OD.planned_dish_id =
PD.planned_dish_id
GROUP BY O.customer_id , C.name, year(target_date),
month(target_date)
```

*Weekly*CompanyCustomerOrdersStatisticsView - *statystyka miesięczna sumy*

wydanej przez firmy oraz ilosc złożonych zamówień zgrupowanych po klientach firmowych

```
CREATE VIEW [WeeklyCompanyCustomerOrdersStatisticsView] AS
```

```
SELECT O.customer_id , C.name,
SUM(PD.price * OD.quantity) AS 'totalPricePaid',
year(target_date) as 'year',
datepart(ww, target_date) as 'week'
FROM Orders O
JOIN CompanyCustomers C on O.customer_id = C.customer_id
INNER JOIN OrderDetails OD on O.order_id = OD.order_id
INNER JOIN PlannedDishes PD on OD.planned_dish_id =
PD.planned_dish_id
GROUP BY O.customer_id , C.name, year(target_date), datepart(ww,
target_date)
```

AmountOfOrdersServedByEmployeesView - *liczba odebranych zamówień przez każdego z pracowników pogrupowane po miesiacach*

```
CREATE VIEW [dbo].[AmountOfOrdersServedByEmployeesView]
AS
SELECT       dbo.Employees.firstname, dbo.Employees.lastname, dbo.Employees.email,
YEAR(dbo.Orders.order_date) AS Year, MONTH(dbo.Orders.order_date)
                  AS Month, COUNT(*) AS Amount
FROM         dbo.Employees INNER JOIN
                  dbo.Orders ON dbo.Employees.employee_id = dbo.Orders.employee_id
WHERE       (dbo.Orders.target_date IS NOT NULL)
GROUP BY dbo.Employees.firstname, dbo.Employees.lastname, dbo.Employees.phone,
dbo.Employees.email, YEAR(dbo.Orders.order_date), MONTH(dbo.Orders.order_date)
GO
```

MonthlyPrivateCustomerOrderDetailsReport - szczegółowy raport miesięczny o zamówieniach klientów indywidualnych

```
CREATE VIEW [MonthlyPrivateCustomerOrderDetailsReport] AS
SELECT O.order_id, O.customer_id, PC.firstname + ' ' + PC.lastname
as 'customerName', PD.name, PD.price * OD.quantity as
dishQuantityCost, PD.price, OD.quantity, target_date,
CONVERT(TIME,target_date) as 'Time', year(target_date) as 'year',
month(target_date) as 'month'
FROM Orders O
INNER JOIN PrivateCustomers PC on O.customer_id = PC.customer_id
```

```
INNER JOIN OrderDetails OD on O.order_id = OD.order_id
INNER JOIN PlannedDishes PD on OD.planned_dish_id =
PD.planned_dish_id
```

WeeklyPrivateCustomerOrderDetailsReport - szczegółowy raport tygodniowy o zamówieniach klientów indywidualnych

```
CREATE VIEW [WeeklyPrivateCustomerOrderDetailsReport] AS
SELECT O.order_id, O.customer_id, PC.firstname + ' ' + PC.lastname
as 'customerName', D.name, PD.price * OD.quantity as
dishQuantityCost, PD.price, OD.quantity, target_date,
CONVERT(TIME,target_date) as 'Time', year(target_date) as 'year',
datepart(ww, target_date) as 'week'
FROM Orders O
INNER JOIN PrivateCustomers PC on O.customer_id = PC.customer_id
INNER JOIN OrderDetails OD on O.order_id = OD.order_id
INNER JOIN PlannedDishes PD on OD.planned_dish_id =
PD.planned_dish_id
Inner join Dishes D on D.dish_id= PD.dish_id
GO
```

MonthlyPrivateCustomerOrdersStatistics - statystyka miesięczna sumy wydanej przez klientów indywidualnych

```
CREATE VIEW [MonthlyPrivateCustomerOrdersStatistics] AS
SELECT O.customer_id , PC.firstname + ' ' + PC.lastname as 'customerName', SUM(price)
AS 'totalPrice', year(target_date) as 'year', month(target_date) as 'month'
FROM Orders O
INNER JOIN PrivateCustomers PC on O.customer_id = PC.customer_id
GROUP BY O.customer_id , PC.firstname + ' ' + PC.lastname, year(target_date),
month(target_date)
```

WeeklyPrivateCustomerOrdersStatistics - statystyka miesięczna sumy wydanej przez klientów indywidualnych

```
CREATE VIEW [WeeklyPrivateCustomerOrdersStatisticsView] AS
SELECT O.customer_id , PC.firstname + ' ' + PC.lastname as 'customerName',
SUM(PD.price * OD.quantity) AS 'totalPrice', year(target_date) as 'year',
datepart(ww, target_date) as 'week'
FROM Orders O
INNER JOIN OrderDetails OD on O.order_id = OD.order_id
INNER JOIN PlannedDishes PD on OD.planned_dish_id = PD.planned_dish_id
INNER JOIN PrivateCustomers PC on O.customer_id = PC.customer_id
GROUP BY O.customer_id , PC.firstname + ' ' + PC.lastname, year(target_date),
datepart(ww, target_date)
```

*Daily*AmountOfCompanyCustomerOrdersView - *liczba wszystkich zamówień firmowych zgrupowane dniami*

```
CREATE VIEW [dbo].[Daily]AmountOfCompanyCustomerOrdersView]
AS
SELECT Day, SUM(Amount) AS Amount
FROM     dbo.AmountOfCompanyCustomerOrdersView
GROUP BY Day
GO
```

MonthlyOrderStatisticsReport - szczegółowy raport miesięczny o zamówieniach wszystkich klientów (ilość na każdego klienta)

*CREATE VIEW [*MonthlyOrderStatisticsReport *] AS*
*SELECT 'Private' as TYPE, * FROM*
[MonthlyPrivateCustomerOrdersStatisticsView]
UNION
*SELECT 'Company' as Type, **
FROM [MonthlyCompanyCustomerOrdersStatisticsView]

WeeklyOrderStatisticsReport - szczegółowy raport tygodniowy o zamówieniach wszystkich klientów (ilość na każdego klienta)

*CREATE VIEW [*WeeklyOrderStatisticsReport *] AS*
*SELECT 'Private' as TYPE, * FROM*
*[*Weekly*PrivateCustomerOrdersStatisticsView]*
UNION
*SELECT 'Company' as Type, **
*FROM [*Weekly*CompanyCustomerOrdersStatisticsView]*

MonthlyOrderDetailsReport - szczegółowy raport miesięczny o zamówieniach wszystkich klientów

CREATE VIEW [MonthlyOrderDetailsReport] AS
*SELECT 'Private' as TYPE, * FROM [MonthlyPrivateCustomerOrderDetailsReport]*
UNION
*SELECT 'Company' as Type, **
FROM [MonthlyCompanyCustomerOrderDetailsReport]

WeeklyOrderDetailsReport - szczegółowy raport miesięczny o zamówieniach wszystkich klientów

CREATE VIEW [WeeklyOrderDetailsReport] AS
*SELECT 'Private' as TYPE, * FROM [WeeklyPrivateCustomerOrderDetailsReport]*
UNION
*SELECT 'Company' as Type, **
FROM [WeeklyCompanyCustomerOrderDetailsReport]

AmountOfOrdersView - liczba wszystkich zamówień

CREATE VIEW [dbo].[AmountOfOrdersView]

```
AS
SELECT COUNT(*) AS Amount, YEAR(order_date) AS year, MONTH(order_date) AS
Month, DATEPART(dw, order_date) AS Day, DATEPART(hh, order_date) AS Hour
FROM     dbo.Orders
GROUP BY YEAR(order_date), MONTH(order_date), DATEPART(dw, order_date),
DATEPART(hh, order_date)
GO
```

DishesView - dane wszystkich dań

```
CREATE VIEW [dbo].[DishesView]
AS
SELECT dbo.Dishes.name, dbo.Dishes.exclusive,  dbo.Categories.name as “Category”
FROM     dbo.Dishes INNER JOIN
              dbo.Categories ON dbo.Dishes.category_id = dbo.Categories.category_id
GO
```

MenuView - aktualne menu

```
CREATE VIEW [dbo].[MenuView]
AS
SELECT dbo.Dishes.name, dbo.Dishes.exclusive, dbo.PlannedDishes.price,
dbo.PlannedDishes.active_from, dbo.PlannedDishes.active_to,  dbo.Categories.name
FROM     dbo.Dishes INNER JOIN
              dbo.PlannedDishes ON dbo.Dishes.dish_id = dbo.PlannedDishes.dish_id INNER
JOIN
              dbo.Categories ON dbo.Dishes.category_id = dbo.Categories.category_id
WHERE  (dbo.PlannedDishes.active_from <= GETDATE()) AND (GETDATE() <=
ISNULL(dbo.PlannedDishes.active_to, GETDATE()))
GO
```

AmountOfPrivateReservationsView - liczba rezerwacji prywatnych

```
CREATE VIEW [dbo].[AmountOfPrivateReservationsView]
AS
SELECT COUNT(*) AS Amount, YEAR(start_time) AS Year, MONTH(start_time) AS Month,
DATEPART(dw, start_time) AS Day, DATEPART(hh, start_time) AS Hour
```

FROM dbo.PrivateReservations GROUP BY MONTH(start_time), YEAR(start_time), DATEPART(dw, start_time), DATEPART(hh, start_time)
GO

AmountOfCompanyReservationsView - liczba rezerwacji firmowych

CREATE VIEW [dbo].[AmountOfCompanyReservationsView]
AS
SELECT COUNT(*) AS Amount, YEAR(start_time) AS Year, MONTH(start_time) AS Month, DATEPART(dw, start_time) AS Day, DATEPART(hh, start_time) AS Hour
FROM dbo.CompanyReservations GROUP BY MONTH(start_time), YEAR(start_time), DATEPART(dw, start_time), DATEPART(hh, start_time)
GO

TablesView - dane o stolikach

CREATE VIEW [dbo].[TablesView]
AS
SELECT seats, is_active
FROM dbo.Tables GO

CompanyReservationWithoutTables - rezerwacje bez stolików dla firm

```
CREATE VIEW [dbo].[CompanyReservationWithoutTables]
AS
SELECT        c.name, r.start_time, r.end_time, d.people_count
FROM          dbo.CompanyReservationTables t
inner join CompanyReservationDetails d on t.reservation_details_id =
d.reservation_details_id
inner join CompanyReservations r on r.company_reservation_id =
d.company_reservation_id
inner join CompanyCustomers c on c.customer_id = r.customer_id
WHERE         (t.table_id  IS  NULL)
```

MonthlyTableReservationsAmount- ile razy dany stolik byl rezerwowany w miesiacu

```
CREATE VIEW [dbo].[MonthlyTableReservationsAmount] AS
select 'CompanyReservation' as TYPE, T.table_id, year(CR.end_time) as 'year',
month(CR.end_time) as 'month', count(CRT.reservation_details_id) as 'HowManyTimes'
from Tables T
left outer join CompanyReservationTables CRT on T.table_id = CRT.table_id
inner join CompanyReservationDetails CRD on CRD.reservation_details_id =
CRT.reservation_details_id
inner join CompanyReservations CR on CR.company_reservation_id = CRD.reservation_id
group by T.table_id, year(CR.end_time), month(CR.end_time)
union
select 'PrivateReservation' as TYPE, T.table_id, year(PR.end_time) as 'year',
month(PR.end_time) as 'month', count(private_reservation_id) as 'HowManyTimes' from
Tables T
left outer join PrivateReservations PR on T.table_id = PR.table_id
group by T.table_id, year(PR.end_time), month(PR.end_time)
```

MonthlyTableReservationsAmount- ile razy dany stolik byl rezerwowany w tygodniu

```
CREATE VIEW [dbo].[WeeklyTableReservationsAmount] AS
select 'CompanyReservation' as TYPE, T.table_id, year(CR.end_time) as 'year',
datepart(ww, CR.end_time) as 'week', count(CRT.reservation_details_id) as
'HowManyTimes' from Tables T
left outer join CompanyReservationTables CRT on T.table_id = CRT.table_id
inner join CompanyReservationDetails CRD on CRD.reservation_details_id =
CRT.reservation_details_id
inner join CompanyReservations CR on CR.company_reservation_id = CRD.reservation_id
group by T.table_id, year(CR.end_time),datepart(ww, CR.end_time)
```

union
select 'PrivateReservation' as TYPE, T.table_id, year(PR.end_time) as 'year', datepart(ww, PR.end_time) as 'week', count(private_reservation_id) as 'HowManyTimes' from Tables T
left outer join PrivateReservations PR on T.table_id = PR.table_id
group by T.table_id, year(PR.end_time),datepart(ww, PR.end_time)

MonthlyDiscountPerCustomerAddedAmount - ilość każdej ze zniżek przydzielonej w miesiacu

CREATE VIEW [dbo].[MonthlyDiscountPerCustomerAddedAmount] AS
select 'TemporaryDiscount' as TYPE, discount_id, discount as 'Discount', year(active_from) as 'Year', month(active_from) as 'Month', count(customer_id) as 'HowManyCustomers' from TemporaryDiscounts
group by discount_id, discount, year(active_from), month(active_from)
union
select 'PermanentDiscount' as TYPE, discount_id, discount as 'Discount', year(active_from) as 'Year', month(active_from) as 'Month', count(customer_id) as 'HowManyCustomers' from PermanentDiscounts
group by discount_id, discount, year(active_from), month(active_from)

WeeklyDiscountPerCustomerAddedAmount - ilość każdej ze zniżek przydzielonej w tygodniu

CREATE VIEW [dbo].[WeeklyDiscountPerCustomerAddedAmount] AS
select 'TemporaryDiscount' as TYPE, discount_id, discount as 'Discount', year(active_from) as 'Year', datepart(ww, active_from) as 'Week', count(customer_id) as 'HowManyCustomers' from TemporaryDiscounts
group by discount_id, discount, year(active_from), datepart(ww, active_from)
union
select 'PermanentDiscount' as TYPE, discount_id, discount as 'Discount', year(active_from) as 'Year', datepart(ww, active_from) as 'Month', count(customer_id) as 'HowManyCustomers' from PermanentDiscounts
group by discount_id, discount, year(active_from), datepart(ww, active_from)

CurrentMenuView- aktualne menu

CREATE VIEW [dbo].[CurrentMenuView] AS
SELECT

```
	d.name as [Nazwa dania],
	description as [Opis],
	price as [Cena],
	c.name as [Kategoria]
FROM
	PlannedDishes pd
	inner join Dishes d on d.dish_id = pd.dish_id
	inner join Categories c on c.category_id = d.category_id
WHERE
	active_from < GETDATE() and active_to IS NULL OR active_to > GETDATE();
GO
```

MenuLastChanges - zmiany w ciagu ostatnich dwóch tygodni w menu

```
CREATE VIEW [dbo].[MenuLastChanges]
AS
SELECT DISTINCT d.name, pd.active_from, pd.active_to, pd.price from PlannedDishes pd
inner join Dishes d on d.dish_id = pd.dish_id
where DATEDIFF(day, active_from, getdate()) <= 14 or (DATEDIFF(day, active_to, getdate())
<= 14 and active_to IS NOT NULL)
```

AllPrivateResevationsView - wszystkie rezerwacje prywatne

```
CREATE view [dbo].[AllPrivateResevationsView] as
SELECT DISTINCT PR.ReservationID,
	pr.time_from,
	pr.time_to,
	rs.status',
	pc.Fristname + ' ' + pc.Lastname as 'assigned_to',
	pr.table_id from PrivateReservations pr
INNER JOIN ReservationStatusHistory rsh on pr.reservation_id = rsh.reservation_id
INNER JOIN ReservationsStatuses rs on rsh.status_id = rs.status_id
INNER JOIN PrivateCustomers pc on pr.customer_id = pc.customer_id
```

CurrentPrivateResevationsView - aktualne rezerwacje prywatne

```
CREATE view [dbo].[CurrentPrivateResevationsView] as
SELECT *
FROM AllPrivateResevationsView
where time_to > GETDATE()
```

TodaysCompanyReservations - dzisiejsze firmowe rezerwacje

```
CREATE VIEW [dbo].[TodaysCompanyReservations] AS
SELECT
	*
FROM
	CompanyReservations cr
WHERE
	NULL not in
	(
	select
		crt.table_id
	from
		CompanyReservationDetails crd
		left join CompanyReservationTables crt on crd.reservaton_details_id =
crt.reservation_details_id
	where
		cr.company_reservation_id = crd.reservation_id
	)
	and (CONVERT(date, start_time) = CONVERT(date, GETDATE()))
GO
```

TodaysPrivateReservations - dzisiejsze rezerwacje indywidualnych klientów

```
CREATE VIEW [dbo].[TodaysPrivateReservations] AS
SELECT
	*
FROM
	PrivateReservations
WHERE
	CONVERT (date, start_time) = CONVERT (date, GETDATE())
	and table_id is not null
GO
```

Procedury

AddCategory

Dodaje nową kategorię.

```
CREATE PROCEDURE [dbo].[AddCategory]
 @CategoryName nvarchar(50)
AS
BEGIN
	BEGIN TRY
		INSERT INTO Categories(name)
		VALUES (@CategoryName);
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg1 nvarchar(1024) = 'Error while inserting
to Categories: '
		+ ERROR_MESSAGE();
		THROW 52000, @errorMsg1, 1;
	END CATCH
END;
```

AddCompany

Dodaje nowego klienta - firmę.

```
CREATE PROCEDURE [dbo].[AddCompany]
 @CompanyName varchar(30),
 @NIP varchar(15),
 @Email varchar(50),
 @Phone varchar(15),
 @Street nvarchar(50),
 @CityID int
AS
BEGIN
 BEGIN TRY
   INSERT INTO Customers (phone, street, city_id) VALUES (@Phone,
@Street, @CityID);
   DECLARE @CustomerID int;
   SELECT @CustomerID = SCOPE_IDENTITY();

   INSERT INTO CompanyCustomers(customer_id, name, nip) VALUES
(@CustomerID, @CompanyName, @NIP);
 END TRY
 BEGIN CATCH
```

```
   DELETE FROM Customers WHERE customer_id = @CustomerID
   DELETE FROM CompanyCustomers WHERE customer_id = @CustomerID
   DECLARE @errorMsg nvarchar(1024) = 'Error while inserting Company: '
   + ERROR_MESSAGE();
   THROW 52000, @errorMsg, 1;
 END CATCH;
END;
```

AddDish

Dodaje nowe danie.

```
CREATE PROCEDURE [dbo].[AddDish]
 @Name nvarchar(50),
 @Category int,
 @Description text,
 @IsSeafood bit
AS
BEGIN
      BEGIN TRY
            INSERT INTO Dishes(name, category_id, description, seafood)
            VALUES (@Name, @Category, @Description, @IsSeafood);
    END TRY
      BEGIN CATCH
            DECLARE @errorMsg1 nvarchar(1024) = 'Error while inserting
to Dishes: '
            + ERROR_MESSAGE();
            THROW 52000, @errorMsg1, 1;
    END CATCH
END;
```

AddPrivateCustomer

Dodaje nowego klienta indywidualnego.

```
CREATE PROCEDURE [dbo].[AddPrivateCustomer]
 @FirstName nvarchar(50),
 @LastName nvarchar(50),
 @Email varchar(50),
 @Phone varchar(15),
 @Street nvarchar(50),
 @CityID int
AS
BEGIN
 BEGIN TRY
```

```
   INSERT INTO Customers (phone, street, city_id) VALUES (@Phone,
@Street, @CityID);
   DECLARE @CustomerID int;
   SELECT @CustomerID = SCOPE_IDENTITY();

   INSERT INTO PrivateCustomers(customer_id, firstname, lastname, email)
VALUES (@CustomerID, @FirstName, @LastName, @Email);
 END TRY
 BEGIN CATCH
   DELETE FROM Customers WHERE customer_id = @CustomerID
   DELETE FROM PrivateCustomers WHERE customer_id = @CustomerID
   DECLARE @errorMsg nvarchar(1024) = 'Error while inserting Private
Customer: '
   + ERROR_MESSAGE();
   THROW 52000, @errorMsg, 1;
 END CATCH;
END;
```

AddEmployee

Dodaje nowego pracownika restauracji.

```
CREATE PROCEDURE [dbo].[AddEmployee]
 @FirstName nvarchar(50),
 @LastName nvarchar(50),
 @Email varchar(50),
 @ReportsTo int
AS
BEGIN
      BEGIN TRY
            INSERT INTO Employees(firstname, lastname, email,
reports_to)
            VALUES (@FirstName, @LastName, @Email, @ReportsTo);
      END TRY
      BEGIN CATCH
            DECLARE @errorMsg1 nvarchar(1024) = 'Error while inserting
to Employees: '
            + ERROR_MESSAGE();
            THROW 52000, @errorMsg1, 1;
      END CATCH
END;
```

CompanyReservationAccept

Zatwierdza firmową rezerwację.

```
CREATE PROCEDURE [dbo].[CompanyReservationAccept]
 @ReservationID int
AS
BEGIN
	IF dbo.GetReservationStatus(@ReservationID, 1) > 1
	BEGIN
		DECLARE @errorMsg3 nvarchar(1024) = 'Cannot accept a
reservation that is already accepted';
		THROW 52000, @errorMsg3, 1;
	END

	IF 0 = (
		SELECT
			MIN(ISNULL(crt.table_id, 0))
		FROM
			CompanyReservationDetails crd
			left join CompanyReservationTables crt on
crd.reservation_details_id = crt.reservation_details_id
		WHERE crd.reservation_id = @ReservationID
	)
	BEGIN
		DECLARE @errorMsg2 nvarchar(1024) = 'Reservation has details
with unassigned tables';
		THROW 52000, @errorMsg2, 1;
	END

	BEGIN TRY
		INSERT INTO ReservationStatusHistory(reservation_id,
is_company, reservation_status_id)
		VALUES (@ReservationID, 1, 2)
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg1 nvarchar(1024) = 'Error while inserting
into ReservationStatusHistory'
		+ ERROR_MESSAGE();
		THROW 52000, @errorMsg1, 1;
	END CATCH
END;
```

CompanyReservationAddDetails

Dodaje szczegóły do rezerwacji firmowej.

```
CREATE PROCEDURE [dbo].[CompanyReservationAddDetails]
 @ReservationID int,
```

```
 @PeopleCount int
AS
BEGIN
	IF (SELECT TOP 1 reservation_status_id FROM
ReservationStatusHistory
	WHERE reservation_id=@ReservationID and is_company=1 ORDER BY
datetime desc) > 1
	BEGIN
		DECLARE @errorMsg2 nvarchar(1024) = 'Cannot add details to
reservation that is accepted';
		THROW 52000, @errorMsg2, 1;
	END

	BEGIN TRY
		INSERT INTO CompanyReservationDetails(reservation_id,
people_count)
		VALUES (@ReservationID, @PeopleCount);
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg1 nvarchar(1024) = 'Error while inserting
to CompanyReservationDetails: '
		+ ERROR_MESSAGE();
		THROW 52000, @errorMsg1, 1;
	END CATCH
END;
```

CompanyReservationAddEmployee

Dodaje imiennie pracownika firmy do rezerwacji firmowej.

```
CREATE PROCEDURE [dbo].[CompanyReservationAddEmployee]
 @ReservationDetailsID int,
 @EmployeeCustomerID int
AS
BEGIN
	BEGIN TRY
		INSERT INTO
CompanyReservationEmployees(reservation_details_id,
employee_customer_id)
		VALUES (@ReservationDetailsID, @EmployeeCustomerID);
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg1 nvarchar(1024) = 'Error while inserting
to CompanyReservationEmployees: '
		+ ERROR_MESSAGE();
		THROW 52000, @errorMsg1, 1;
```

```
	END CATCH
END;
```

CompanyReservationAssignTable

Przypisuje stolik do rezerwacji firmowej (szczegółów rezerwacji).

```
CREATE PROCEDURE [dbo].[CompanyReservationAssignTable]
 @ReservationDetailsID int,
 @TableID int
AS
BEGIN
	DECLARE @ReservationID int = (SELECT reservation_id FROM
CompanyReservationDetails WHERE
reservation_details_id=@ReservationDetailsID)
	DECLARE @Start datetime = (SELECT start_time FROM
CompanyReservations WHERE company_reservation_id=@ReservationID)
	DECLARE @End datetime = (SELECT end_time FROM CompanyReservations
WHERE company_reservation_id=@ReservationID)
	DECLARE @People int = (SELECT people_count FROM
CompanyReservationDetails WHERE
reservation_details_id=@ReservationDetailsID)

	IF dbo.IsTableAvailable(@TableID, @Start, @End)=0
	BEGIN
		DECLARE @errorMsg2 nvarchar(1024) = 'The table is not
available';
		THROW 52000, @errorMsg2, 1;
	END

	IF (SELECT seats FROM Tables WHERE table_id=@TableID)<@People
	BEGIN
		DECLARE @errorMsg3 nvarchar(1024) = 'The table does not have
enough seats';
		THROW 52000, @errorMsg3, 1;
	END

	BEGIN TRY
		INSERT INTO CompanyReservationTables(reservation_details_id,
table_id)
		VALUES (@ReservationDetailsID, @TableID);
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg1 nvarchar(1024) = 'Error while inserting
to CompanyReservationDetails: '
		+ ERROR_MESSAGE();
```

```
            THROW 52000, @errorMsg1, 1;
      END CATCH
END;
```

CompanyReservationPlace

Tworzy nową, póki co pustą, rezerwację firmową.

```
CREATE PROCEDURE [dbo].[CompanyReservationPlace]
 @CustomerID int,
 @Start datetime,
 @HowManyHours int
AS
BEGIN
      IF dbo.GetCustomerType(@CustomerID)!='company'
      BEGIN
            DECLARE @errorMsg3 nvarchar(1024) = 'Only company customer
can make a company reservation';
            THROW 52000, @errorMsg3, 1;
      END

      BEGIN TRY
            DECLARE @End datetime = DATEADD(HOUR, @HowManyHours, @Start)

            INSERT INTO CompanyReservations(customer_id, start_time,
end_time)
            VALUES (@CustomerID, @Start, @End)

            DECLARE @ReservationID int = SCOPE_IDENTITY();
            INSERT INTO ReservationStatusHistory(reservation_id,
is_company, reservation_status_id)
            VALUES (@ReservationID, 1, 1)
      END TRY
      BEGIN CATCH
            DECLARE @errorMsg1 nvarchar(1024) = 'Error while inserting
to CompanyReservations: '
            + ERROR_MESSAGE();
            THROW 52000, @errorMsg1, 1;
      END CATCH
END;
```

GrantPermanentDiscount

Daje klientowi permanentną zniżkę.

```
CREATE PROCEDURE [dbo].[GrantPermanentDiscount]
```

```
 @CustomerID int
AS
BEGIN
 IF dbo.CanCustomerGetPermanentDiscount(@CustomerID) = 1
 BEGIN
   BEGIN TRY
     DECLARE @DiscountValue real = (SELECT value FROM DiscountParams
WHERE param_id='R1')/100.0
     DECLARE @ActiveFrom datetime = GETDATE();
       DECLARE @DiscountID int = (SELECT max(discount_id) FROM
DiscountParamHistory)

     INSERT INTO PermanentDiscounts(customer_id, discount_id, discount,
active_from)
     VALUES (
             @CustomerID,
             @DiscountID,
             @DiscountValue,
             @ActiveFrom
       );
   END TRY
   BEGIN CATCH
     DECLARE @errorMsg1 nvarchar(1024) = 'Error while inserting to
PermanentDiscounts: '
     + ERROR_MESSAGE();
     THROW 52000, @errorMsg1, 1;
   END CATCH
 END
 ELSE
 BEGIN
   DECLARE @errorMsg2 nvarchar(1024) = CONCAT('Customer ', @CustomerID,
' is not eligible for Permanent Discount');
   THROW 52000, @errorMsg2, 1;
 END
END;
```

GrantTemporaryDiscount

Daje klientowi czasową zniżkę.

```
CREATE PROCEDURE [dbo].[GrantTemporaryDiscount]
 @CustomerID int
AS
BEGIN
 IF dbo.CanCustomerGetTemporaryDiscount(@CustomerID) = 1
 BEGIN
```

```
   BEGIN TRY
     DECLARE @DiscountValue real = (SELECT value FROM DiscountParams
WHERE param_id='R2')/100.0
     DECLARE @DiscountPeriod int = (SELECT value FROM DiscountParams
WHERE param_id='D1')
     DECLARE @ActiveFrom datetime = GETDATE();
     DECLARE @ActiveTo datetime = DATEADD(DAY, @DiscountPeriod,
@ActiveFrom);
       DECLARE @DiscountID int = (SELECT max(discount_id) FROM
DiscountParamHistory)

     INSERT INTO TemporaryDiscounts(customer_id, discount_id, discount,
active_from, active_to)
     VALUES (
             @CustomerID,
             @DiscountID,
             @DiscountValue,
             @ActiveFrom,
             @ActiveTo
       );
   END TRY
   BEGIN CATCH
     DECLARE @errorMsg1 nvarchar(1024) = 'Error while inserting to
TemporaryDiscounts: '
     + ERROR_MESSAGE();
     THROW 52000, @errorMsg1, 1;
   END CATCH
 END
 ELSE
 BEGIN
   DECLARE @errorMsg2 nvarchar(1024) = CONCAT('Customer ', @CustomerID,
' is not eligible for Temporary Discount');
   THROW 52000, @errorMsg2, 1;
 END
END;
```

OrderAccept

Zatwierdza zamówienie.

```
CREATE PROCEDURE [dbo].[OrderAccept]
 @OrderID int,
 @EmployeeID int
AS
BEGIN
      IF (SELECT TOP 1 order_status_id FROM OrderStatusHistory WHERE
```

```
order_id=@OrderID ORDER BY datetime desc) > 1
	BEGIN
		DECLARE @errorMsg1 nvarchar(1024) = 'Cannot accept an Order
that is accepted';
		THROW 52000, @errorMsg1, 1;
	END

	BEGIN TRY
		UPDATE Orders SET employee_id = @EmployeeID WHERE
order_id=@OrderID
		INSERT INTO OrderStatusHistory(order_id, order_status_id)
		VALUES (@OrderID, 2);
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg2 nvarchar(1024) = 'Error while accepting
Order: '
		+ ERROR_MESSAGE();
		THROW 52000, @errorMsg1, 1;
	END CATCH
END;
```

OrderAddDetails

Dodaje szczegóły do zamówienia (pozycje z menu).

```
CREATE PROCEDURE [dbo].[OrderAddDetails]
 @OrderID int,
 @PlannedDishID int,
 @Quantity int
AS
BEGIN
	IF dbo.GetOrderStatus(@OrderID) > 1
	BEGIN
		DECLARE @errorMsg1 nvarchar(1024) = 'Cannot add details to
an Order that is accepted';
		THROW 52000, @errorMsg1, 1;
	END

	DECLARE @OrderDate datetime =(SELECT order_date FROM Orders WHERE
order_id=@OrderID);
	IF dbo.IsPlannedDishAvailable(@PlannedDishID, @OrderDate)=0
	BEGIN
		DECLARE @errorMsg2 nvarchar(1024) = 'Item not on the current
menu';
		THROW 52000, @errorMsg2, 1;
	END
```

```
      IF dbo.isSeafood(@PlannedDishID)=1
      BEGIN
     DECLARE @OrderTargetDate datetime = (SELECT target_date FROM Orders
WHERE order_id=@OrderID);

     IF DATEPART(WEEKDAY, @OrderTargetDate) <5 OR DATEPART(WEEKDAY,
@OrderTargetDate) >7
     BEGIN
       DECLARE @errorMsg3 nvarchar(1024) = 'Seafood can only be ordered
for a day between Thursday and Saturday';
       THROW 52000, @errorMsg3, 1;
     END;

       DECLARE @MondayDate datetime = DATEADD(DAY, -DATEPART(WEEKDAY,
@OrderTargetDate)+2, @OrderTargetDate);
     IF (SELECT order_date FROM Orders WHERE
order_id=@OrderID)>@MondayDate
     BEGIN
       DECLARE @errorMsg4 nvarchar(1024) = 'Seafood can only be ordered
before Monday preceding order date';
       THROW 52000, @errorMsg4, 1;
     END;
   END;

      BEGIN TRY
            INSERT INTO OrderDetails(order_id, planned_dish_id,
quantity)
            VALUES (@OrderID, @PlannedDishID, @Quantity);
      END TRY
      BEGIN CATCH
            DECLARE @errorMsg5 nvarchar(1024) = 'Error while inserting
to OrderDetails: '
            + ERROR_MESSAGE();
            THROW 52000, @errorMsg5, 1;
      END CATCH
END;
```

OrderFull

Składa pełne zamówienie ze szczegółami.

```
CREATE PROCEDURE [dbo].[OrderFull]
 @CustomerID int,
```

```
 @TargetDate datetime,
 @Values OrderDetailsList READONLY,
 @OUT int OUTPUT
AS
BEGIN
      BEGIN TRY
       DECLARE @OrderID int;
     EXEC dbo.OrderPlace @CustomerID, @TargetDate, @OrderID OUTPUT;

       DECLARE OrderCursor CURSOR FOR SELECT planned_dish_id, quantity
FROM @Values
       OPEN OrderCursor
       DECLARE @PlannedDishID int, @Quantity int
       FETCH NEXT FROM OrderCursor INTO @PlannedDishID, @Quantity
       WHILE @@FETCH_STATUS=0
       BEGIN
             EXEC dbo.OrderAddDetails @OrderID, @PlannedDishID,
@Quantity
             FETCH NEXT FROM OrderCursor INTO @PlannedDishID, @Quantity
       END
       CLOSE OrderCursor
       DEALLOCATE OrderCursor

       SELECT @OUT = @OrderID
      END TRY
      BEGIN CATCH
            DELETE FROM Orders WHERE order_id=@OrderID

            DECLARE @errorMsg1 nvarchar(1024) = 'Error while ordering: '
            + ERROR_MESSAGE();
            THROW 52000, @errorMsg1, 1;
      END CATCH
END;
```

OrderPlace

Dodaje nowe, póki co puste, zamówienie.

```
CREATE PROCEDURE [dbo].[OrderPlace]
 @CustomerID int,
 @TargetDate datetime
AS
BEGIN
```

```
      BEGIN TRY
            INSERT INTO Orders(customer_id, order_date, target_date)
            VALUES (@CustomerID, GETDATE(), @TargetDate);
            DECLARE @OrderID int = SCOPE_IDENTITY();

            INSERT INTO OrderStatusHistory(order_id, order_status_id)
            VALUES (@OrderID, 1);
      END TRY
      BEGIN CATCH
            DECLARE @errorMsg1 nvarchar(1024) = 'Error while inserting
to Orders: '
            + ERROR_MESSAGE();
            THROW 52000, @errorMsg1, 1;
      END CATCH
END;
```

PlanDish

Planuje danie, to znaczy określa za ile i kiedy będzie dostępny w menu.

```
CREATE PROCEDURE [dbo].[PlanDish]
 @DishID int,
 @Price money,
 @Start datetime,
 @End datetime
AS
BEGIN
   BEGIN TRY
         INSERT INTO PlannedDishes(dish_id, price, active_from,
active_to)
         VALUES (@DishID, @Price, @Start, @End);
   END TRY
   BEGIN CATCH
     DECLARE @errorMsg1 nvarchar(1024) = 'Error while inserting to
PlannedDishes: '
     + ERROR_MESSAGE();
     THROW 52000, @errorMsg1, 1;
   END CATCH
END;
```

PrivateReservationAccept

Zatwierdza rezerwację klienta indywidualnego, przydzielając stolik.

```
CREATE PROCEDURE [dbo].[PrivateReservationAccept]
 @ReservationID int,
```

```
 @TableID int
AS
BEGIN
	IF dbo.GetReservationStatus(@ReservationID, 0) > 1
	BEGIN
		DECLARE @errorMsg3 nvarchar(1024) = 'Cannot accept a
reservation that is already accepted';
		THROW 52000, @errorMsg3, 1;
	END

	DECLARE @Start datetime = (SELECT start_time FROM
PrivateReservations WHERE private_reservation_id=@ReservationID)
	DECLARE @End datetime = (SELECT end_time FROM PrivateReservations
WHERE private_reservation_id=@ReservationID)

	IF dbo.IsTableAvailable(@TableID, @Start, @End)=0
	BEGIN
		DECLARE @errorMsg2 nvarchar(1024) = 'The table is not
available';
		THROW 52000, @errorMsg2, 1;
	END

	BEGIN TRY
		UPDATE PrivateReservations SET table_id = @TableID WHERE
private_reservation_id=@ReservationID;
		INSERT INTO ReservationStatusHistory(reservation_id,
is_company, reservation_status_id)
		VALUES (@ReservationID, 0, 2)
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg1 nvarchar(1024) = 'Error while updating
PrivateReservations'
		+ ERROR_MESSAGE();
		THROW 52000, @errorMsg1, 1;
	END CATCH
END;
```

PrivateReservationAndOrder

Składa rezerwację i zamówienie jedocześnie.

```
CREATE PROCEDURE [dbo].[PrivateReservationAndOrder]
 @CustomerID int,
 @Values OrderDetailsList READONLY,
 @TargetDate datetime,
```

```
 @PeopleCount int,
 @HowManyHours int,
 @OUT int OUTPUT
AS
BEGIN
      BEGIN TRY
       DECLARE @OrderID int;
     EXEC dbo.OrderFull @CustomerId, @TargetDate, @Values, @OrderID
OUTPUT;

       DECLARE @ReservationID int;
       EXEC PrivateReservationPlace @OrderID, @PeopleCount,
@HowManyHours, @ReservationID OUTPUT;

       SELECT @OUT = @OrderID
      END TRY
      BEGIN CATCH
            DELETE FROM Orders WHERE order_id=@OrderID

            DECLARE @errorMsg1 nvarchar(1024) = 'Error while placing
order with reservation: '
            + ERROR_MESSAGE();
            THROW 52000, @errorMsg1, 1;
      END CATCH
END;
```

PrivateReservationPlace

Tworzy nową rezerwację dla klienta indywidualnego (związaną z zamówieniem).

```
CREATE PROCEDURE [dbo].[PrivateReservationPlace]
 @OrderID int,
 @PeopleCount int,
 @HowManyHours int
AS
BEGIN
      DECLARE @CustomerID int = (SELECT customer_id FROM Orders WHERE
order_id=@OrderID)

      IF dbo.GetCustomerType(@CustomerID)!='private'
      BEGIN
            DECLARE @errorMsg3 nvarchar(1024) = 'Only private customer
can make a private reservation';
            THROW 52000, @errorMsg3, 1;
      END
```

```
	IF dbo.CanMakeReservation(@CustomerID, @OrderID)=0
	BEGIN
		DECLARE @errorMsg2 nvarchar(1024) = 'Cannot make reservation
- requirements not met';
		THROW 52000, @errorMsg2, 1;
	END

	BEGIN TRY
		DECLARE @OrderTargetDate datetime = (SELECT target_date FROM
Orders WHERE order_id=@OrderID)
		DECLARE @EndTime datetime = DATEADD(HOUR, @HowManyHours,
@OrderTargetDate)

		INSERT INTO PrivateReservations(customer_id, order_id,
people_count, start_time, end_time)
		VALUES (@CustomerID, @OrderID, @PeopleCount,
@OrderTargetDate, @EndTime)

		DECLARE @ReservationID int = SCOPE_IDENTITY();
		INSERT INTO ReservationStatusHistory(reservation_id,
is_company, reservation_status_id)
		VALUES (@ReservationID, 0, 1)

	END TRY
	BEGIN CATCH
		DECLARE @errorMsg1 nvarchar(1024) = 'Error while inserting
to PrivateReservations: '
		+ ERROR_MESSAGE();
		THROW 52000, @errorMsg1, 1;
	END CATCH
END;
```

UpdateDiscountParam

Zmienia wartość parametru dotyczącego zniżek.

```
CREATE PROCEDURE [dbo].[UpdateDiscountParam]
 @ParamID varchar(10),
 @Value int
AS
BEGIN
 BEGIN TRY
   UPDATE DiscountParamHistory SET active_to=GETDATE() WHERE (active_to
IS NULL) AND param_id=@ParamID;
```

```
   UPDATE DiscountParams SET value=@Value WHERE param_id=@ParamID;
   INSERT INTO DiscountParamHistory(param_id, value, active_from,
active_to) VALUES (@ParamID, @Value, GETDATE(), NULL);
 END TRY
 BEGIN CATCH

   DECLARE @errorMsg2 nvarchar(1024) = 'Error while inserting Discount
Param: '
   + ERROR_MESSAGE();
   THROW 52000, @errorMsg2, 1;
 END CATCH
END;
```

Funkcje

CanCustomerGetPermanentDiscount

Zwraca czy klientowi można przyznać permanentną zniżkę.

```
CREATE FUNCTION [dbo].[CanCustomerGetPermanentDiscount](@CustomerID int)
RETURNS bit --returns whether customer get permanent discount on current
conditions
AS
BEGIN
 IF dbo.GetCustomerType(@CustomerID) != 'private'
 BEGIN
   RETURN 0;
 END
  IF EXISTS (select customer_id from PermanentDiscounts where
customer_id=@CustomerID)
  BEGIN
   RETURN 0;
  END
 DECLARE @RequiredOrderCount int = dbo.GetDiscountParamValue('Z1',
GETDATE());
 DECLARE @RequiredAmountPerOrder int = dbo.GetDiscountParamValue('K1',
GETDATE());

 IF @RequiredOrderCount <= (
       SELECT count(*)
       FROM Orders
       WHERE
            customer_id = @CustomerID
            and dbo.GetOrderTotalAmount(order_id) >=
@RequiredAmountPerOrder
 )
```

```
 BEGIN
   RETURN 1;
 END
 RETURN 0;
END;
```

CanCustomerGetTemporaryDiscount

Zwraca czy klientowi można przyznać tymczasową zniżkę.

```
CREATE FUNCTION [dbo].[CanCustomerGetTemporaryDiscount](@CustomerID int)
RETURNS bit --returns whether customer get temporary discount on current
conditions
AS
BEGIN
 IF dbo.GetCustomerType(@CustomerID) != 'private'
  BEGIN
   RETURN 0;
 END
 DECLARE @RequiredTotalAmount int = dbo.GetDiscountParamValue('K2',
GETDATE());
 DECLARE @EndOfLastTemporaryDiscount datetime =
 ISNULL((SELECT active_to FROM TemporaryDiscounts WHERE
customer_id=@CustomerID), '1999-01-01')
 IF @RequiredTotalAmount <= (
   SELECT SUM(dbo.GetOrderTotalAmount(order_id))
   FROM
      Orders
   WHERE
      customer_id = @CustomerID
      and order_date >= @EndOfLastTemporaryDiscount
 )
 BEGIN
   RETURN 1;
 END
 RETURN 0;
END;
```

CanMakeReservation

Zwraca czy klient (indywidualny) może dokonać rezerwacji.

```
CREATE FUNCTION [dbo].[CanMakeReservation] (@CustomerID int, @OrderID
int)
```

```
RETURNS bit
AS
BEGIN
	IF dbo.GetCustomerType(@CustomerID) = 'company'
	BEGIN
		RETURN 1;
	END

	DECLARE @MinOrderCount int =
	(SELECT value from ReservationParams where param_id='WK')

	DECLARE @CustomersOrderCount int =
	(SELECT count(*) from Orders where customer_id=@CustomerID)

	DECLARE @MinOrderTotal int =
	(SELECT value from ReservationParams where param_id='WZ')

	DECLARE @OrderTotal int =
	dbo.GetOrderTotalAmount(@OrderID)

	IF( @MinOrderCount<=@CustomersOrderCount and
@MinOrderTotal<=@OrderTotal)
	BEGIN
		RETURN 1;
	END
	RETURN 0;
END;
```

GetAvailableTable

Zwraca najmniejszy wolny stół w danym przedziale czasu, który pomieści daną ilość osób lub NULL jeśli nie ma takiego stołu.

```
CREATE FUNCTION [dbo].[GetAvailableTable](@Start datetime, @End
datetime, @People int)
RETURNS int
AS
BEGIN
	RETURN (
		SELECT TOP 1 table_id
		FROM Tables
		WHERE dbo.IsTableAvailable(table_id, @Start, @End)=1
			and seats>=@People
		ORDER BY seats asc
```

```
            )
END;
```

GetCustomersDiscount

Zwraca wartość zniżki danego klienta w danym momencie.

```
CREATE FUNCTION [dbo].[GetCustomersDiscount](@CustomerID int, @Date
datetime)
RETURNS real --returns customers discount value for a given date - the
highest available is chosen
AS
BEGIN

 DECLARE @PermanentDiscount real =
 ISNULL(
 (SELECT
      discount
 FROM
      PermanentDiscounts
 WHERE
      customer_id = @CustomerID
      and (active_from <= @Date)
 ),
 0)
 DECLARE @TemporaryDiscount real =
 ISNULL(
 (SELECT
      discount
 FROM
      TemporaryDiscounts
 WHERE
      customer_id = @CustomerID
      and (active_from <= @Date)
      and (active_to is null or active_to >= @Date)
 ),
 0)
 IF (@TemporaryDiscount >= @PermanentDiscount)
 BEGIN
      RETURN @TemporaryDiscount;
 END
 RETURN @PermanentDiscount;
END;
```

GetCustomerType

Zwraca czy klient jest indywidualnym czy firmowym.

```
CREATE FUNCTION [dbo].[GetCustomerType](@CustomerID int)
RETURNS varchar(10)
AS
BEGIN
 IF @CustomerID in (
      SELECT customer_id
      FROM PrivateCustomers
      )
 BEGIN
      RETURN 'private';
 END
  IF @CustomerID in (
      SELECT customer_id
      FROM CompanyCustomers
      )
 BEGIN
      RETURN 'company';
 END
 RETURN NULL;
END;
```

GetDiscountParamValue

Zwraca wartość parametru zniżek dla danej daty.

```
CREATE FUNCTION [dbo].[GetDiscountParamValue] (@ParamID varchar(10),
@Date datetime NULL)
RETURNS int
AS
BEGIN
 IF(@Date is NULL)
 BEGIN
      SET @Date = GETDATE()
 END

 RETURN (
   SELECT Value
   FROM DiscountParamHistory
   WHERE param_id = @ParamID
     AND (active_to IS NULL OR active_to >= @Date)
       AND (active_from <= @Date)
 )
```

```
END;
```

GetOrderStatus

Zwraca aktualny (najnowszy) status zamówienia.

```
CREATE FUNCTION [dbo].[GetOrderStatus](@OrderID int)
RETURNS int
AS
BEGIN
      RETURN (SELECT TOP 1 order_status_id FROM OrderStatusHistory
      WHERE order_id=@OrderID ORDER BY datetime desc)
END;
```

GetOrderTotalAmount

Zwraca całkowitą wartość zamówienia, uwzględniając zniżki.

```
CREATE FUNCTION [dbo].[GetOrderTotalAmount] (@OrderID int)
RETURNS money
AS
BEGIN
 RETURN (
   SELECT
      SUM(od.quantity * pd.price *
      (1-dbo.GetCustomersDiscount(o.customer_id, o.order_date))
      ) AS TotalAmount
   FROM
         Orders o
         INNER JOIN OrderDetails od ON o.order_id = od.order_id
         INNER JOIN PlannedDishes pd ON pd.planned_dish_id =
od.planned_dish_id
   WHERE o.order_id = @OrderID
 )
END;
```

GetReservationStatus

Zwraca aktualny (najnowszy) status rezerwacji.

```
CREATE FUNCTION [dbo].[GetReservationStatus](@ReservationID int,
```

```
@IsCompany bit)
RETURNS int
AS
BEGIN
      RETURN (SELECT TOP 1 reservation_status_id FROM
ReservationStatusHistory
      WHERE reservation_id=@ReservationID and is_company=@IsCompany
ORDER BY datetime desc)
END;
```

IsMenuFresh

Zwraca czy menu spełnia zasadę, że co najmniej połowa pozycji menu zmieniana jest co najmniej raz na dwa tygodnie.

```
CREATE FUNCTION [dbo].[IsMenuFresh] (@Date datetime)
RETURNS bit
AS
BEGIN
 DECLARE @ItemsOlderThan14Days int = (
       SELECT count(dish_id)
       FROM PlannedDishes
       WHERE
            (active_to>@Date or active_to is NULL)
            and DATEDIFF(DAY, @Date, active_from) >= 14
            and active_from<GETDATE()
 )
 DECLARE @MenuLength int = (
       SELECT count(*)
      FROM PlannedDishes
      WHERE active_from < @Date and (active_to IS NULL OR active_to >
@Date)
 )
 IF @MenuLength >= 2*@ItemsOlderThan14Days
 BEGIN
      RETURN 1
 END
 RETURN 0
END;
```

IsPlannedDishAvailable

Zwraca czy dane danie z menu jest dostępne w danym momencie.

```
CREATE FUNCTION [dbo].[IsPlannedDishAvailable] (@PlannedDishID int,
@Date datetime)
RETURNS bit
AS
BEGIN
	IF @PlannedDishID in (
		SELECT planned_dish_id
		FROM PlannedDishes
		WHERE active_from < @Date and (active_to IS NULL OR
active_to > @Date)
	)
	BEGIN
		RETURN 1
	END
	RETURN 0
END;
```

IsSeafood

Zwraca czy dane danie z menu zawiera owoce morza i obowiązują specjalne zasady zamawiania go.

```
CREATE FUNCTION [dbo].[IsSeafood] (@PlannedDishID int)
RETURNS bit
AS
BEGIN
	IF 'Seafood' = (SELECT c.name
	FROM
		Dishes d
		inner join PlannedDishes pd on pd.dish_id=d.dish_id
		inner join Categories c on d.category_id = c.category_id
	WHERE planned_dish_id=@PlannedDishID)

	BEGIN
		RETURN 1;
	END
	RETURN 0;

END;
```

IsTableAvailable

Zwraca czy stolik w danym przedziale czasowym jest dostępny.

```
CREATE FUNCTION [dbo].[IsTableAvailable] (@TableID int, @Start datetime,
@End datetime)
```

```
RETURNS bit
AS
BEGIN
 IF 0 = ISNULL((
        SELECT is_active
        FROM Tables
        WHERE table_id = @TableID),
        0)
  BEGIN
      RETURN 0;
  END
 DECLARE @HasPrivateReservation bit =
 ISNULL((SELECT
      table_id
 FROM PrivateReservations
 WHERE
      table_id=@TableID
      and start_time between @Start and @End
      and end_time between @Start and @End
      and @Start between start_time and end_time),0)

 DECLARE @HasCompanyReservation bit =
 ISNULL((SELECT
      table_id
 FROM
      CompanyReservationTables crt
      inner join CompanyReservationDetails crd on
crd.reservation_details_id = crt.reservation_details_id
      inner join CompanyReservations cr on cr.company_reservation_id =
crd.reservation_id
 WHERE
      table_id=@TableID
      and start_time between @Start and @End
      and end_time between @Start and @End
      and @Start between start_time and end_time),0)

      IF(@HasPrivateReservation=0 and @HasCompanyReservation=0)
      BEGIN
            RETURN 1;
      END
      RETURN 0;
END;
```

FullyDeleteCompanyReservationDetails

Przy usuwaniu z `CompanyReservationDetails` usuwa także wszystkie związane rekordy.

```
CREATE TRIGGER [dbo].[FullyDeleteCompanyReservationDetails]
ON [dbo].[CompanyReservationDetails]
INSTEAD OF DELETE
AS
BEGIN
      DECLARE @ReservationDetailsID int = (SELECT reservation_details_id
FROM DELETED)

      DELETE FROM CompanyReservationEmployees WHERE
reservation_details_id=@ReservationDetailsID
      DELETE FROM CompanyReservationTables WHERE
reservation_details_id=@ReservationDetailsID
      DELETE FROM CompanyReservationDetails WHERE reservation_details_id
=@ReservationDetailsID
END;
```

CheckCompanyReservationEmployeeAmount

Zapobiega dodaniu imiennie do grupy rezerwacyjnej większej ilości pracowników, niż określona ilość osób w grupie rezerwacyjnej.

```
CREATE TRIGGER [dbo].[CheckCompanyReservationEmployeeAmount]
ON [dbo].[CompanyReservationEmployees]
AFTER INSERT
AS
BEGIN
 DECLARE @ReservationDetailsID int=(SELECT reservation_details_id FROM
INSERTED);
 DECLARE @EmployeeCount int=(SELECT count(employee_customer_id) FROM
CompanyReservationEmployees WHERE
reservation_details_id=@ReservationDetailsID);
 DECLARE @PeopleCountInReservationDetails int=(SELECT people_count FROM
CompanyReservationDetails WHERE
reservation_details_id=@ReservationDetailsID);

 IF @EmployeeCount > @PeopleCountInReservationDetails
 BEGIN
   DECLARE @ErrorMsg nvarchar(100) = 'Cannot add more employees';
   RAISERROR(@ErrorMsg, 1, 1);
   ROLLBACK TRANSACTION
 END;
END;
```

CheckIfReservationCanBeModified

Zapobiega dodawaniu nowych pracowników imiennie do rezerwacji, która jest zatwierdzona.

```
CREATE TRIGGER [dbo].[CheckIfReservationCanBeModified]
ON [dbo].[CompanyReservationEmployees]
AFTER INSERT
AS
BEGIN
 DECLARE @ReservationDetailsID int=(SELECT reservation_details_id FROM
INSERTED);

      IF dbo.GetReservationStatus(
            (select reservation_id
            from CompanyReservationDetails
            where reservation_details_id=@ReservationDetailsID),
            1) > 1
      BEGIN
               DECLARE @ErrorMsg nvarchar(100) = 'This reservation
cannot be modified';
               RAISERROR(@ErrorMsg, 1, 1);
               ROLLBACK TRANSACTION
      END
END;
```

FullyDeleteCompanyReservation

Usuwa całość rezerwacji firmowej przy usuwaniu z `CompanyReservationDetails`.

```
CREATE TRIGGER [dbo].[FullyDeleteCompanyReservation]
ON [dbo].[CompanyReservations]
INSTEAD OF DELETE
AS
BEGIN
      DECLARE @ReservationID int = (SELECT company_reservation_id FROM
DELETED)

      DELETE FROM ReservationStatusHistory WHERE
reservation_id=@ReservationID and is_company=1
      DELETE FROM CompanyReservationDetails WHERE
reservation_id=@ReservationID
      DELETE FROM CompanyReservations WHERE company_reservation_id
=@ReservationID
END;
```

FullyDeleteOrder

Usuwa całość zamówienia.

```
CREATE TRIGGER [dbo].[FullyDeleteOrder]
ON [dbo].[Orders]
INSTEAD OF DELETE
AS
BEGIN
      DECLARE @OrderID int = (SELECT order_id FROM DELETED)

      DELETE FROM OrderDetails WHERE order_id = @OrderID
      DELETE FROM OrderStatusHistory WHERE order_id = @OrderID
      DELETE FROM PrivateReservations WHERE order_id = @OrderID
      DELETE FROM Orders WHERE order_id = @OrderID
END;
```

GrantDiscount

Przyznaje discount klientowi, jeśli spełnia wymagania i go nie ma.

```
CREATE TRIGGER [dbo].[GrantDiscount]
ON [dbo].[Orders]
AFTER INSERT
AS
BEGIN
 DECLARE @CustomerID int=(SELECT customer_id FROM INSERTED);
 IF dbo.CanCustomerGetTemporaryDiscount(@CustomerID) = 1
 BEGIN
   EXEC GrantTemporaryDiscount @CustomerID
 END;
 IF dbo.CanCustomerGetPermanentDiscount(@CustomerID) = 1
 BEGIN
   EXEC GrantPermanentDiscount @CustomerID
 END;
END;
```

FullyDeletePrivateReservation

Usuwa całość rezerwacji indywidualnej.

```
CREATE TRIGGER [dbo].[FullyDeletePrivateReservation]
ON [dbo].[PrivateReservations]
INSTEAD OF DELETE
AS
BEGIN
      DECLARE @ReservationID int = (SELECT private_reservation_id FROM
DELETED)

      DELETE FROM ReservationStatusHistory WHERE
reservation_id=@ReservationID and is_company=0
      DELETE FROM PrivateReservations WHERE private_reservation_id
=@ReservationID
END;
```

CheckForReservationID

Upewnia się, że rekord w `ReservationStatusHistory` opisuje rezerwację firmowej lub indywidualną.

```
CREATE TRIGGER [dbo].[CheckForReservationID]
ON [dbo].[ReservationStatusHistory]
AFTER INSERT
AS
BEGIN
      DECLARE @ReservationID int = (SELECT reservation_id from INSERTED)
      DECLARE @IsCompany int = (SELECT is_company from INSERTED)
 IF @IsCompany=1
 BEGIN
      IF NOT EXISTS (select * from CompanyReservations where
company_reservation_id=@ReservationID)
      BEGIN
          RAISERROR('Cannot find a corresponding reservation in
CompanyReservations', 1, 1)
            ROLLBACK TRANSACTION
      END
 END
 IF @IsCompany=0
 BEGIN
      IF NOT EXISTS (select * from PrivateReservations where
private_reservation_id=@ReservationID)
      BEGIN
          RAISERROR('Cannot find a corresponding reservation in
PrivateReservations', 1, 1)
            ROLLBACK TRANSACTION
      END
 END
```

```
END;
```

Indeksy

Orders

```
CREATE NONCLUSTERED INDEX OrdersCustomerIDIndex ON Orders(customer_id);

CREATE NONCLUSTERED INDEX OrdersDatesIDIndex ON Orders(order_date,
target_date);
```

PrivateReservations

```
CREATE NONCLUSTERED INDEX PrivateReservationsCustomerIDIndex ON
PrivateReservations(customer_id);

CREATE NONCLUSTERED INDEX PrivateReservationsOrderIDIndex ON
PrivateReservations(order_id);

CREATE NONCLUSTERED INDEX PrivateReservationsDatesIDIndex ON
PrivateReservations(starts_at, ends_at);
```

CompanyReservations

```
CREATE NONCLUSTERED INDEX CompanyReservationsCompanyIDIndex] ON
CompanyReservations(customer_id);

CREATE NONCLUSTERED INDEX CompanyReservationsDatesIDIndex ON
CompanyReservations(starts_at, ends_at);
```

CompanyCustomers

```
CREATE UNIQUE NONCLUSTERED INDEX CompanyNIPIndex ON
CompanyCustomers(nip);
```

Categories

```
CREATE UNIQUE NONCLUSTERED INDEX CategoriesIndex ON
Categories(CategoryName);
```

PlannedDishes

```
CREATE NONCLUSTERED INDEX PlannedDishesDatesIDIndex ON
PlannedDishes(active_from, active_to);

CREATE NONCLUSTERED INDEX PlannedDishDishIDIndex ON
PlannedDishes(dish_id);
```

PermanentDiscounts

```
CREATE UNIQUE NONCLUSTERED INDEX PermanentDiscountsDateIndex ON
PermanentDiscounts(active_from);
```

TemporaryDiscounts

```
CREATE UNIQUE NONCLUSTERED INDEX TemporaryDiscountsDateIndex ON
TemporaryDiscounts(active_from, active_to);

CREATE UNIQUE NONCLUSTERED INDEX TemporaryDiscountsCustomersIndex ON
TemporaryDiscounts(customer_id);
```

DiscountParamHistory

```
CREATE UNIQUE NONCLUSTERED INDEX DiscountParamHistoryParamIDIndex ON
DiscountParamHistorys(param_id);

CREATE UNIQUE NONCLUSTERED INDEX DiscountParamHistoryDateIndex ON
DiscountParamHistorys(active_from, active_to);
```

Uprawnienia

Admin

```
CREATE ROLE Admin AUTHORIZATION dbo
GRANT all to Admin
```

Employee

```
CREATE ROLE Employee AUTHORIZATION dbo
GRANT EXECUTE ON AddCompany to Employee
GRANT EXECUTE ON AddPrivateCustomer to Employee
GRANT EXECUTE ON CompanyReservationAccept to Employee
GRANT EXECUTE ON CompanyReservationAssignTable to Employee
GRANT EXECUTE ON CompanyReservationPlaceto Employee
GRANT EXECUTE ON PrivateReservationPlace to Employee
GRANT EXECUTE ON OrderAcceptto Employee
GRANT EXECUTE ON AddCategory to Employee
GRANT EXECUTE ON AddNewCity to Employee
GRANT EXECUTE ON addNewCountry to Employee
GRANT EXECUTE ON PlaceOrder to Employee
GRANT EXECUTE ON ChangeOrderStatus to Employee
```

Manager

```
CREATE ROLE Manager AUTHORIZATION dbo
GRANT EXECUTE ON AddCompany to Manager
GRANT EXECUTE ON AddPrivateCustomer to Manager
GRANT EXECUTE ON CompanyReservationAccept to Manager
GRANT EXECUTE ON CompanyReservationAssignTable to Manager
GRANT EXECUTE ON CompanyReservationPlace to Manager
GRANT EXECUTE ON PrivateReservationPlace to Manager
GRANT EXECUTE ON OrderAcceptto Manager
GRANT EXECUTE ON PrivateReservationWithOrder to Manager
GRANT EXECUTE ON PrivateReservationPlace to Manager
GRANT EXECUTE ON AddCategory to Manager
GRANT EXECUTE ON PlanDish to Manager
GRANT EXECUTE ON AddNewCity to Manager
GRANT EXECUTE ON addNewCountry to Manager
GRANT EXECUTE ON PlaceOrder to Manager
GRANT EXECUTE ON ChangeOrderStatus to Manager
```

CompanyCustomer

```
CREATE ROLE CompanyCustomer AUTHORIZATION dbo
GRANT EXECUTE ON CompanyReservationPlace to CompanyCustomer
GRANT EXECUTE ON CompanyReservationAddDetails to CompanyCustomer
```

```
GRANT EXECUTE ON CompanyReservationAddEmployees to CompanyCustomer
GRANT EXECUTE ON OrderFull to CompanyCustomer

GRANT SELECT ON CurrentMenuView to CompanyCustomer
```

PrivateCustomer

```
CREATE ROLE PrivateCustomer AUTHORIZATION dbo
GRANT EXECUTE ON PrivateReservationAndOrder to PrivateCustomer
GRANT EXECUTE ON OrderFull to PrivateCustomer

GRANT SELECT ON CurrentMenuView to CompanyCustomer
```