SPRAWOZDANIE PROJEKTU	
Informatyka – Data Science	Karolina Maruszak

1. Analiza wymagań.

Diagram użycia:

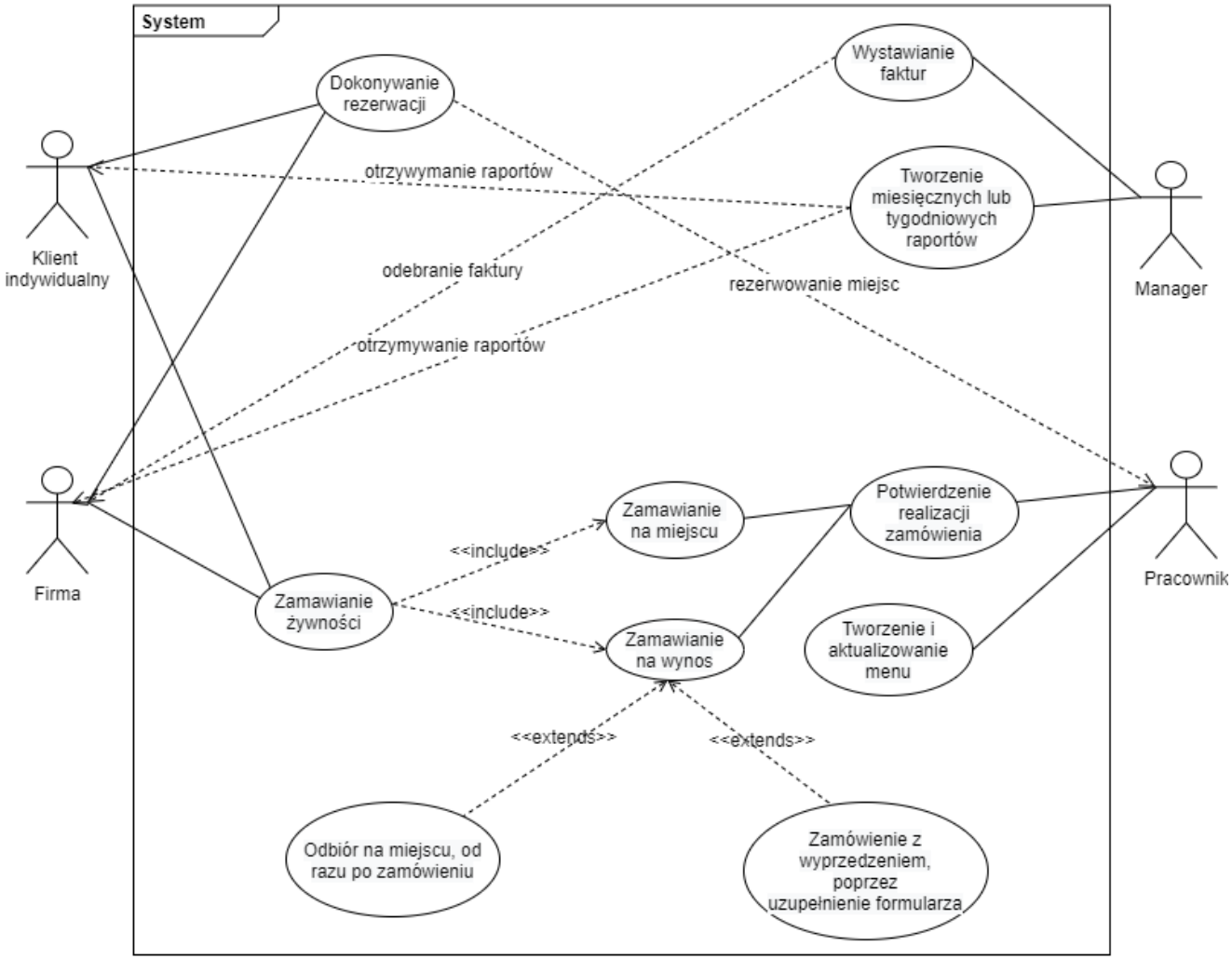


Diagram użycia

Dziedzina problemu:

System dla firmy świadczącej usługi gastronomiczne dla klientów indywidualnych oraz firm.

Główne funkcje systemu:

- Formularz WWW który umożliwia dokonać zamówienia na wynos lub na miejscu z wyprzedzeniem.
- Korzystając z formularza, możliwość wyboru preferowanej godziny i daty dla odbioru zamówienia.
- Ze względu na COVID -19, ograniczona liczba miejsc – zmienna w czasie.
- System powinien posiadać funkcję rezerwacji stolika dla co najmniej dwóch osób.
- Funkcjonalność wystawienia faktury dla zamówienia lub faktury zbiorczej (dla firm).
- Menu ustalane z dziennym wyprzedzeniem, aktualizowane przez pracownika.
- Rozbudowywanie menu dla poszczególnych dni w tygodniu (w czwartek, piątek oraz sobotę istnieje możliwość wcześniejszego zamówienia dań zawierających owoce morza).
- Zautomatyzowany system usuwania produktów z menu w przypadku brakujących produktów.
- Formularz internetowy pozwalający klientowi indywidualnemu rezerwacji stolika wraz z dokonaniem zamówienia.
- Internetowy formularz dla firm z funkcjami rezerwacji stolików na firmę i/lub rezerwacji stolików dla pracowników firmy (wtedy rezerwacja jest imienna).
- Wybór opcji płatności przy rezerwacji stolika wraz z dokonaniem zamówienia. Płatność przed lub po zamówieniu oraz zabezpieczenie na kwotę minimalną 50 PLN dla zamówienia.
- Funkcjonalność umożliwiająca akceptacje dla rezerwacji klienta indywidualnego wraz z informacją zwrotną zawierającą potwierdzenie zamówienia oraz stolika.
- System realizujący rabaty dla klientów indywidualnych oraz dla firm
- Funkcjonalność generowania raportów miesięcznych i tygodniowych dla klienta indywidualnego oraz firm.

Scenariusze:

Scenariusz do rezerwacji dla klienta indywidualnego:

1. Klient indywidualny wybiera sposób rezerwacji stolika.
2. Klient dokonuje wyboru płatności.
3. Oczekiwanie na potwierdzenie rezerwacji przez pracownika.
4. Rezerwacja stolika dla klienta indywidualnego.
5. Klient indywidualny dokonuje zamówienia żywności.

Scenariusz alternatywny:

1.1. W przypadku rezerwacji przez formularz, złóż jednocześnie zamówienie.

1.2. W przypadku dokonywania rezerwacji w inny sposób przejdź do punktu 3.

2.1. Nie wybrano opcji płatności – wybierz odpowiednią opcję.
2.2. Rezerwacja jest dokonywana dla co najmniej dwóch osób.

3.1. Dane są niekompletne – uzupełnij brakujące pola.
3.2. Wypełnij formularz ponownie.

4.1. W przypadku zmiany decyzji anuluj rezerwację stolika.
5.1. Nie wybrano opcji jakiego typu jest zamówienie – wybierz odpowiednią opcję.
5.1.1. W przypadku zamówienia na wynos - nie wypełniono formularza poprawnie.
5.2. Minimalna wartość zamówienia aby dokonać rezerwacji stolika to 50 PLN.

Scenariusz do rezerwacji dla firmy:

1. Firma wybiera sposób rezerwacji stolika.
2. Firma dokonuje wyboru płatności.
3. Oczekiwanie potwierdzenie rezerwacji stolika dla firmy przez pracownika.
4. Rezerwacja stolika dla firmy przez pracownika.
5. Firma dokonuje zamówienia żywności.

Scenariusz alternatywny :

1.1. W przypadku rezerwacji przez formularz wybierz odpowiednią opcję rezerwacji stolików na firmę.

1.1.1. Rezerwacja jest dokonywana dla indywidualnych pracowników(imiennie).

1.1.2. Rezerwacja jest dokonywana dla całej firmy.

1.2. W przypadku rezerwacji w inny sposób przejdź do punktu 3.

2.1.Nie wybrano żadnej z opcji płatności.
2.2. Wykonaj ten krok raz jeszcze.

3.1. Dane niekompletne – uzupełnij brakujące pola.
3.2. Wypełnij formularz ponownie.

4.1. W przypadku zmiany decyzji anuluj rezerwację stolika.

5.1. Nie wybrano opcji jakiego typu jest zamówienie – wybierz odpowiednią opcję.
5.1.1. W przypadku zamówienia na wynos nie wypełniono formularza poprawnie.
5.2. Minimalna wartość zamówienia aby dokonać rezerwacji stolika to 50 PLN.

Scenariusz zamawiania żywności przez klienta indywidualnego:

1. Klient wybiera z dostępnego menu zamówienie.
2. Klient dokonuje zamówienia.
3. Klient czeka na odbiór zamówienia.
4. Klient odbiera zamówienie.

Scenariusz alternatywny:

1.1. W przypadku problemów z systemem podczas korzystania z formularza WWW przy zamówieniu na wynos, wykonaj telefon pod numer podany na stronie.

2.1. Dokonaj zamówienia.

2.1.1. Dokonaj zamówienia na miejscu.
2.1.2. Dokonaj zamówienia na wynos.

2.1.2.a. Skorzystaj z formularza WWW.
2.1.2.b. Wybierz datę i godzinę odbioru.
2.1.2.c. Poczekaj na potwierdzenie zamówienia.

4.1. Sprawdź czy zamówienie się zgadza.
4.2. Firma odbiera większą ilość posiłków w porze lunchu lub jako catering.
4.3. Brak dostaw.

Scenariusz do wystawiania faktur oraz raportowania przez Managera:

1. Manager dokonuje wystawienia faktury dla klienta indywidualnego lub firmy.
2. Manager tworzy tygodniowy lub miesięczny raport dla klienta indywidualnego lub firmy.
3. Manager wysyła raport do klienta indywidualnego lub firmy.

Scenariusz alternatywny:

1.1. Wprowadzone dane do faktury są błędne.
    1.1.1. Sprawdź poprawność danych.
    1.1.2. Wypełnij dokument ponownie.

1.2. Nie zapisano wprowadzonych danych.
    1.2.1. Zapisz dane i wystaw fakturę ponownie.

1.3. Wybierz sposób wystawienia faktury dla firmy.
    1.3.1. Firma decyduje się na fakturę elektroniczną.
    1.3.2. Firma decyduje się na fakturę tradycyjną.

1.4. Otrzymanie raportu przez firmę.

2.1. Wprowadzone dane do raportu są błędne.
    2.1.1. Sprawdź poprawność danych.
    2.1.2. Wypełnij dokument ponownie.

2.2. Dokonaj wyboru typu raportu.
    2.2.1. Nie wybrano odpowiedniej opcji.
    2.2.2. Spróbuj ponownie.

3.1. Raport nie został wysłany.
    3.1.1. Ponów próbę wysłania raportu.

3.2. Brak informacji o kliencie lub firmie na jakie dane ma zostać wysłany raport.
    3.2.1. Odszukaj w systemie/ rachunkach odpowiednie dane.
    3.2.2. Ponów próbę przesłania raportu.

3.3. Otrzymanie raportu przez klienta indywidualnego / firmę.

Scenariusz dla czynności wykonywanych przez pracownika:

1. Pracownik tworzy lub aktualizuje menu.
2. Pracownik dokonuje potwierdzenie zamówienia stolika dla klienta lub firmy.

Scenariusz alternatywny :

1.1. Nie dokonano aktualizacji menu.
    1.1.1. Wprowadziłeś błędne informacje.
    1.1.2. Wprowadziłeś puste pole.
    1.1.3. Zaktualizowałeś złe pole w menu.

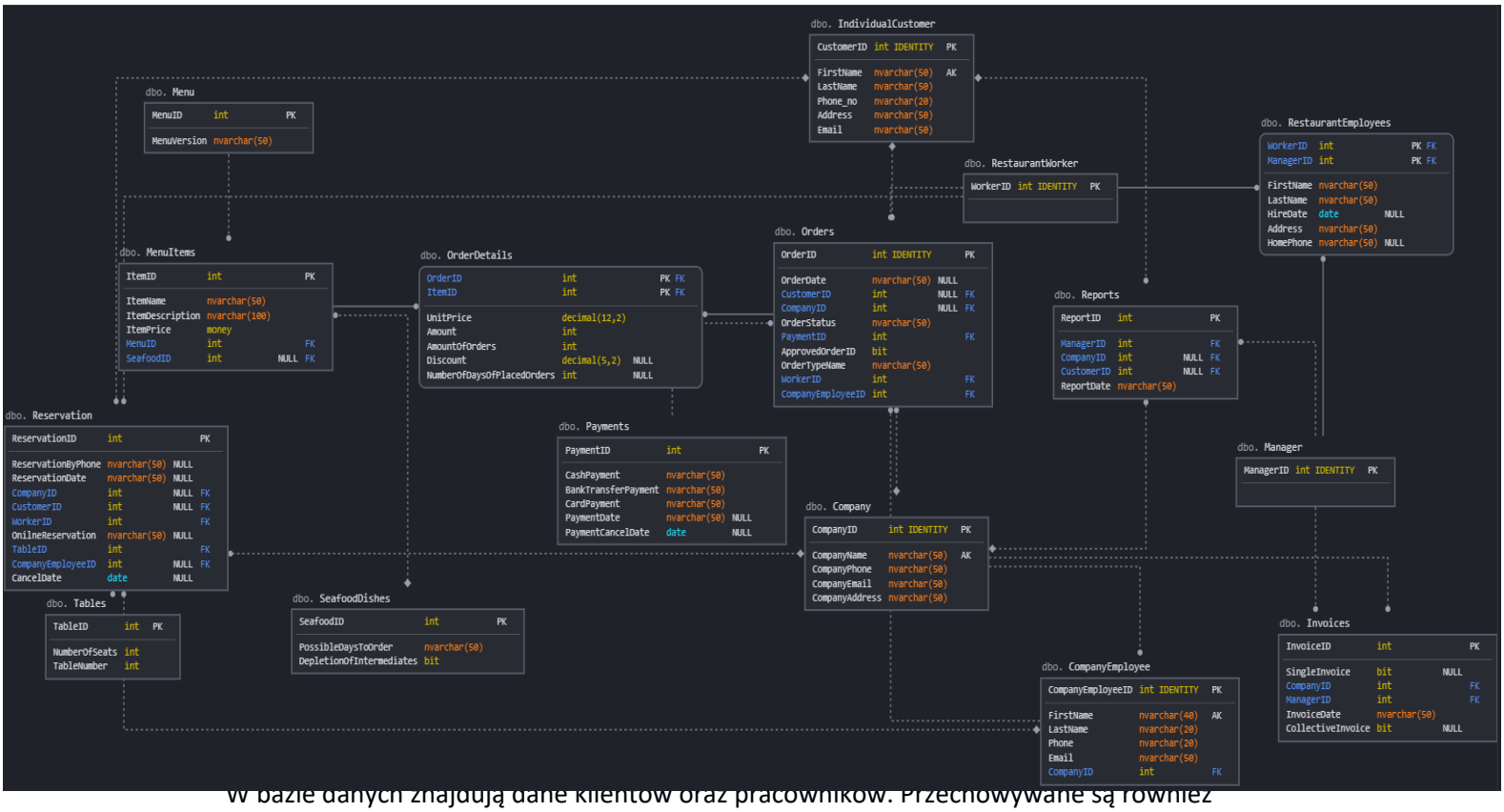
2.1. Nadesłany przez klienta formularz jest błędny.
    2.1.1. Skontaktuj się z klientem w celu doprecyzowania formularza.
    2.1.2. W przeciwnym przypadku odrzuć zamówienie.

2.2. W przypadku dokonania rezygnacji przez klienta, anuluj rezerwację stolika lub zamówienia.

2.3. Potwierdź rezerwację stolika lub zamówienia.

2.4. Dane zostały zapisane w systemie.

2. Schemat bazy danych.



W bazie danych znajdują dane klientów oraz pracowników. Przechowywane są również informacje dotyczące m.in. rezerwacji stolików, zamówień, menu, sposobów płatności, raportów, faktur oraz dostępnych rabatów.

4. Opis tabel.

[Tables]:

```
CREATE TABLE [Tables](
     TableID int NOT NULL ,
     NumberOfSeats int NOT NULL,
     TableNumber int UNIQUE  NOT NULL,
     CONSTRAINT PK_tables PRIMARY KEY CLUSTERED (TableID ASC),
     CONSTRAINT Table_Number_Tables CHECK (TableNumber > 0),
     CONSTRAINT Number_Of_Seats_Tables CHECK (NumberOfSeats >= 2)
 )
```

Tabela [Tables] przechowuje informacje dotyczące możliwych do rezerwacji stolików: ID odpowiedniego stolika (TableID), liczbę dostępnych miejsc przy stoliku (NumberOfSeats) oraz

numer odpowiedniego stolika, który jest unikatowy (TableNumber). Żadna z kolumn nie przyjmuje wartości NULL. Ponadto numer stolika musi być liczbą naturalną większą od zera (stoliki numerujemy od 1) oraz minimalna liczba miejsc przy stoliku, wynosi 2. Maksymalna liczba stolików w restauracji może wynosić 25.

Company:

```
CREATE TABLE Company (
    CompanyID     int IDENTITY (1, 1) NOT NULL,
    CompanyName    nvarchar(50) NOT NULL ,
    CompanyPhone   nvarchar(50) NOT NULL ,
    CompanyEmail  nvarchar(50) NOT NULL ,
    CompanyAddress nvarchar(50) NOT NULL ,

    CONSTRAINT PK_Supplier PRIMARY KEY CLUSTERED (CompanyID ASC),
    CONSTRAINT AK1_Supplier_CompanyName UNIQUE NONCLUSTERED (CompanyName ASC),
    CONSTRAINT Check_Email_Com CHECK (CompanyEmail LIKE '%_@_%._%')
)
```

Tabela Company przechowuje informacje dotyczące danych firm, będących klientami firmy gastronomicznej: ID danej firmy (CompanyID), nazwę firmy (CompanyName), numer telefonu odpowiedniej firmy (CompanyPhone), adres e-mail (CompanyEmail) oraz adres siedziby firmy (CompanyAddress). Żadne z pól nie może przyjmować wartości NULL, ponieważ zarówno nazwa firmy jak i numer telefonu czy adres e-mail, są konieczne przy dokonywaniu przez firmę rezerwacji. Ponadto e-mail musi mieć postać odpowiednio podaną w CHECKU, czyli odpowiednimi adresami będą np.: ‘firma@gmail.com’, ‘firma@op.pl’ itp.

IndividualCustomer:

```
CREATE TABLE IndividualCustomer (
    CustomerID int IDENTITY (1, 1) NOT NULL,
    FirstName nvarchar(50) NOT NULL ,
    LastName nvarchar(50) NOT NULL ,
    Phone_no  nvarchar(20) NOT NULL CONSTRAINT [DF_IndividualCustomer_Phone_no]
DEFAULT ((0)) ,
    [Address] nvarchar(50) NULL ,
    Email    nvarchar(50) NOT NULL ,

 CONSTRAINT PK_Product PRIMARY KEY CLUSTERED (CustomerID ASC),
 CONSTRAINT AK1_Product_SupplierId_ProductName UNIQUE NONCLUSTERED (FirstName ASC),
 CONSTRAINT Check_Email_IC CHECK (Email LIKE '%_@_%._%')
)
```

Tabela IndividualCustomer przechowuje dane klientów indywidualnych, korzystających z usług firmy gastronomicznej: ID klienta (CustomerID), imię oraz nazwisko (FirstName, Lastname), numer telefonu klienta (Phone_no), adres zamieszkania ([Address]) oraz adres e-mail (Email). Wszystkie pola nie mogą być NULL-ami poza adresem zamieszkania, ponieważ nie jest on konieczny przy rezerwacji i nie jest wymagany do podania. Tak jak w przypadku tabeli Company, adres e-mail musi mieć postać np.: ‘klient@gmail.com’ , ‘klient@cokolwiek.org’ itp.

Kolumny w których należy podać imię, nazwisko czy adres e-mail są typu nvarchar.

RestaurantWorker:

```
CREATE TABLE RestaurantWorker (
    WorkerID int IDENTITY (1, 1) NOT NULL ,
    CONSTRAINT PK_workers PRIMARY KEY CLUSTERED (WorkerID ASC)
)
```

Tabela RestaurantWorker przechowuje ID pracowników firmy gastronomicznej, zajmujących się dokonywaniem rezerwacji, zamówieniami, płatnościami.

Reservation:

```
CREATE TABLE Reservation (
    ReservationID     int NOT NULL,
    ReservationByPhone nvarchar(50) NULL ,
    ReservationDate   nvarchar(50) NOT NULL,
    CompanyID         int NULL ,
    CustomerID         int NULL ,
    WorkerID          int NOT NULL ,
    OnilneReservation  nvarchar(50) NULL ,
    TableID           int NOT NULL ,
    CompanyEmployeeID int NULL,
    CancelDate date NULL

    CONSTRAINT PK_reservation PRIMARY KEY CLUSTERED (ReservationID ASC),
    CONSTRAINT FK_company_res FOREIGN KEY (CompanyID)  REFERENCES
    Company(CompanyID),
          CONSTRAINT FK_individual_customer_res FOREIGN KEY (CustomerID)
    REFERENCES IndividualCustomer(CustomerID),
          CONSTRAINT FK_restaurant_worker_res FOREIGN KEY (WorkerID)  REFERENCES
    RestaurantWorker(WorkerID),
          CONSTRAINT FK_table_res FOREIGN KEY (TableID)  REFERENCES [Tables]
    (TableID),
          CONSTRAINT FK_company_em FOREIGN KEY (CompanyEmployeeID)  REFERENCES
    CompanyEmployee( CompanyEmployeeID)
)
```

Tabela Reservation przechowuje informacje dotyczące rezerwacji stolików dokonywanej przez: klienta indywidualnego, firmę lub pojedynczego pracownika z odpowiedniej firmy. Kolumny jakie znajdujemy w tabeli to: ID rezerwacji (ReservationID), dwie opcje rezerwowania stolika, online przez dostępny formularz na stronie (OnlineReservation) lub poprzez rezerwację telefoniczną, a więc konieczne jest podanie numeru telefonu klienta (ReservationByPhone), data rezerwacji (ReservationDate), odpowiednie ID klienta (CompanyID lub CustomerID lub CompanyEmployeeID), ID pracownika który obsługuję daną rezerwację (WorkerID), ID stolika na który dokonywana jest rezerwacja (TableID) oraz ewentualna data anulowania(CancelDate).
Kluczami obcymi są zatem: CustomerID, CompanyID, CompanyEmployeeID, WorkerID, TableID.
Wartości NULL przyjmują wszystkie kolumny oprócz: WorkerID, TableID oraz ReservationDate.

Indeksy:

```
CREATE NONCLUSTERED INDEX fkIdx_company_res ON Reservation (
  CompanyID ASC
 )


CREATE NONCLUSTERED INDEX fkIdx_individual_customer_res ON Reservation (
  CustomerID ASC
 )


CREATE NONCLUSTERED INDEX fkIdx_restaurant_worker_res ON Reservation (
  WorkerID ASC
 )


CREATE NONCLUSTERED INDEX fkIdx_table_res ON Reservation (
  TableID ASC
 )


CREATE NONCLUSTERED INDEX fkIdx_company_em ON Reservation (
  CompanyEmployeeID ASC
 )
```

Menu:

```
CREATE TABLE Menu (
     MenuID     int NOT NULL ,
     MenuVersion nvarchar(50) NOT NULL ,

     CONSTRAINT PK_menu PRIMARY KEY CLUSTERED (MenuID ASC)
)
```

Tabela Menu przechowuje informacje dotyczące ID menu (MenuID) i odpowiednią wersję menu (MenuVersion), która przedstawiana jest w postaci daty ostatniej modyfikacji menu.

SeaFoodDishes:

```
CREATE TABLE SeaFoodDishes (
     SeaFoodID               int NOT NULL ,
     PossibleDaysToOrder     nvarchar(50) NOT NULL ,
     DepletionOfIntermediates bit NOT NULL ,

     CONSTRAINT PK_SeaFoodDishes PRIMARY KEY CLUSTERED (SeaFoodID ASC)

)
```

Tabela SeaFoodDishes przechowuje informacje dotyczące przysmaków owoców morza, które pojawiają się w menu tylko w określone dni. Kolumny jakie tabela zawiera to: ID odpowiedniego dania (SeaFoodID), tak jak wyżej wspomniane – określone dni w które można zamawiać owoce

morza (PossibleDaysToOrder) oraz informacje dotyczące wyczerpania składników (`DepletionOfIntermediates`). Żadne z pól nie może przyjmować wartości NULL.

MenuItems:

```
CREATE TABLE MenuItems (
     ItemID        int NOT NULL ,
     ItemName       nvarchar(50) NOT NULL ,
     ItemDescription nvarchar(100) NOT NULL ,
     ItemPrice       money NOT NULL ,
     MenuID          int NOT NULL ,
     SeaFoodID      int  NULL ,


     CONSTRAINT PK_menuitems PRIMARY KEY CLUSTERED (ItemID ASC),
     CONSTRAINT FK_menu_id FOREIGN KEY (MenuID)  REFERENCES Menu(MenuID),
     CONSTRAINT FK_sea_food_id FOREIGN KEY (SeaFoodID)  REFERENCES
SeaFoodDishes(SeaFoodID),
     CONSTRAINT Check_Price_MI CHECK (ItemPrice > 0)

)
```

Tabela MenuItems przechowuje informacje dotyczące „wnętrza menu”, czyli całego spisu dostępnych potraw. Elementami tabeli są: ID odpowieniej opcji z menu (ItemID), nazwa przysmaku (ItemName), opis (ItemDescription), pojedyncza cena produktu (ItemPrice), ID menu z którego produkt pochodzi (MenuID) oraz ewentualne ID dania z owocami morza (SeaFoodID).
Kluczami obcymi są MenuID oraz SeaFoodID (może przyjmować wartość NULL, ponieważ nie jest dostępne w menu przez cały czas). Ponadto cena nie może być ani mniejsza ani równa 0.

Indeksy:

```
CREATE NONCLUSTERED INDEX fkIdx_menu_id ON MenuItems (
  MenuID ASC
 )



CREATE NONCLUSTERED INDEX fkIdx_sea_food_id ON MenuItems (
  SeafoodID ASC
 )
```

Payments:

```
CREATE TABLE Payments (
 PaymentID         int NOT NULL ,
 CashPayment        bit  NULL ,
 BankTransferPayment bit  NULL ,
 CardPayment       bit   NULL ,
 PaymentDate       nvarchar(50) NULL,
 PaymentCancelDate date NULL


 CONSTRAINT PK_payments PRIMARY KEY CLUSTERED (PaymentID ASC),
```

```
 CONSTRAINT payment_check CHECK((CashPayment='FALSE' AND BankTransferPayment='FALSE'
AND CardPayment='TRUE') OR (CashPayment='FALSE' AND BankTransferPayment='TRUE'
 AND CardPayment='FALSE') OR
 (CashPayment='TRUE' AND BankTransferPayment='FALSE' AND CardPayment='FALSE'))
  )
```

Tabela Payments przechowuje informacje na temat sposobów płatności: ID płatności (PaymentID), płatność gotówką (CashPayment), płatność przelewem bankowym (BankTransferPayment), płatność kartą (CardPayment), data płatności (PaymentDate) oraz ewentualna data anulowania płatności (PaymentCancelDate). Rodzaje płatności są typu bit, a więc mogą przyjmować jedną z dwóch wartości: true lub false, z tym że w danym momencie tylko jeden ze sposobów może być true (ponieważ nie da się w jednym momencie dokonać płatności dwoma lub trzema metodami na raz).

Orders:

```
CREATE TABLE Orders (
 OrderID          int IDENTITY (1, 1) NOT NULL ,
 CompanyEmployeeID int  NULL ,
 OrderDate         nvarchar(50) NULL ,
 CustomerID        int NULL ,
 CompanyID         int NULL ,
 OrderStatus       nvarchar(50) NOT NULL ,
 PaymentID         int NOT NULL ,
 ApprovedOrderID   bit NOT NULL ,
 OrderTypeName     nvarchar(50) NOT NULL ,
 WorkerID         int NOT NULL ,


 CONSTRAINT PK_Order PRIMARY KEY CLUSTERED (OrderId ASC),
 CONSTRAINT FK_customer_id FOREIGN KEY (CustomerID)  REFERENCES
IndividualCustomer(CustomerID),
 CONSTRAINT FK_company_em_id FOREIGN KEY (CompanyEmployeeID)  REFERENCES
CompanyEmployee(CompanyEmployeeID),
 CONSTRAINT FK_company_id_orders FOREIGN KEY (CompanyID)  REFERENCES
Company(CompanyID),
 CONSTRAINT FK_payment_id FOREIGN KEY (PaymentID)  REFERENCES Payments(PaymentID),
 CONSTRAINT FK_worker_id FOREIGN KEY (WorkerID)  REFERENCES
RestaurantWorker(WorkerID)
  )
```

Tabela Orders przechowuje informacje na temat zamówień złożonych przez klientów: ID zamówienia (OrderID), ID odpowiednich klientów (CompanyEmployeeID, CustomerID, CompanyID), datę zamówienia (OrderDate), status zamówienia (OrderStatus), ID płatności (PaymentID), informacje czy zamówienie zostało zatwierdzone (ApprovedOrderID), typ zamówienia z czego dostępne jest zamówienie online lub telefoniczne (OrderTypeName) oraz ID pracownika obsługującego konkretne zamówienie (WorkerID). Kluczami obcymi są: CompanyEmployeeID, CustomerID, CompanyID, WorkerID, PaymentID. ApprovedOrderID przyjmuje wartość true lub false. Z kolei OrderStatus może przyjmować wartości: ‘completed’ oraz ‘accepted’.

Indeksy:

```
CREATE NONCLUSTERED INDEX fkIdx_customer_id ON Orders (
  CustomerID ASC
 )


CREATE NONCLUSTERED INDEX fkIdx_company_em_id ON Orders (
  CompanyEmployeeID ASC
 )


CREATE NONCLUSTERED INDEX fkIdx_company_id_orders ON Orders (
  CompanyID ASC
 )


CREATE NONCLUSTERED INDEX fkIdx_payment_id ON Orders (
  PaymentID ASC
 )


CREATE NONCLUSTERED INDEX fkIdx_worker_id ON Orders (
  WorkerID ASC
```

OrderDetails:

```
CREATE TABLE OrderDetails (
 OrderID    int NOT NULL ,
 ItemID    int NOT NULL ,
 Amount     int NOT NULL ,
 UnitPrice  decimal(12,2) NOT NULL ,
 AmountOfOrders int NOT NULL,
 Discount decimal(5,2) NULL ,
 NumberOfDaysOfPlacedOrders int NULL,


 CONSTRAINT PK_OrderItem PRIMARY KEY CLUSTERED (OrderID ASC, ItemID ASC),
 CONSTRAINT FK_order_id FOREIGN KEY (OrderID)  REFERENCES Orders(OrderID),
 CONSTRAINT FK_item_id FOREIGN KEY (ItemID)  REFERENCES MenuItems(ItemID),
 CONSTRAINT Check_Amount_OD CHECK (Amount > 0),
 CONSTRAINT Check_UnitProce_OD CHECK (UnitPrice > 0),
 CONSTRAINT Check_Amount_Of_Orders_OD CHECK (AmountOfOrders > 0)
  )
```

Tabela OrderDetails przechowuje szczegółowe informacje dotyczące zamówień opisywanych w tabeli Orders oraz zamówień konkretnych produktów i ich ilości z tabeli MenuItems. Zatem kluczami głównymi a zarazem obcymi są OrderID oraz ItemID. Pozostałymi kolumnami w tabeli OrderDetails są: ilość odpowiednich produktów (Amount), cena za odpowiednią ilość produktów (UnitPrice), ilość takich zamówień (AmountOfOrders), zniżka na dane zamówienie (Discount), oraz liczba dni przez które dane zamówienie zostało składane (NumberOfDaysOfPlacedOrders).

Zniżka liczona jest poprzez funkcje, które są zaprezentowane w późniejszej części tego sprawozdania. Ostatnia kolumna (NumberOfDaysOfPlacedOrders) wykorzystywana jest do obliczenia zniżki ze względu na liczbę dni. Ponadto zniżka może przyjmować wartość NULL. Co więcej oczywiste jest, że zarówno ilość odpowiednich produktów jak i ilość zamówień muszą być większe od 0.

Indeksy:

```
CREATE NONCLUSTERED INDEX fkIdx_order_id ON OrderDetails (
  OrderID ASC
 )


CREATE NONCLUSTERED INDEX fkIdx_item_id ON OrderDetails (
  ItemID ASC
 )
```

Manager:

```
CREATE TABLE Manager (
 ManagerID int IDENTITY (1, 1) NOT NULL ,

 CONSTRAINT PK_manager PRIMARY KEY CLUSTERED (ManagerID ASC)
  )
```

Tabela Manager przechowuje ID managerów firmy gastronomicznej, którzy zajmują się wstawianiem faktur i pisaniem odpowiednich raportów.

RestaurantEmployees:

```
CREATE TABLE RestaurantEmployees (
 WorkerID int NOT NULL ,
 ManagerID int  NOT NULL ,
 FirstName nvarchar(50) NOT NULL ,
 LastName  nvarchar(50) NOT NULL ,
 HireDate  date NULL ,
 [Address]   nvarchar(50) NOT NULL ,
 HomePhone nvarchar(50)  NULL ,


 CONSTRAINT PK_restaurantemployee PRIMARY KEY CLUSTERED (WorkerID ASC, ManagerID
ASC),
 CONSTRAINT FK_worker_id FOREIGN KEY (WorkerID)  REFERENCES
RestaurantWorker(WorkerID),
 CONSTRAINT FK_manager_id FOREIGN KEY (ManagerID)  REFERENCES Manager(ManagerID)
   )
```

Tabela RestaurantEmployees przechowuje dane pracowników firmy gastronomicznej, a więc zwykłych pracowników oraz managerów. Zatem kluczami obcymi a zarazem głównymi są WorkerID

(z tabeli RestaurantWorker) oraz ManagerID (z tabeli Manager). Tabela RestaurantEmployees posiada takie kolumny jak: imię oraz nazwisko pracownika (FirstName, LastName), datę zatrudnienia, która może przyjmować wartość NULL (HireDate), adres zamieszkania pracownika ([Address]) oraz numer telefonu domowego, który również może przyjmować wartość NULL (HomePhone).

Indeksy:

```
CREATE NONCLUSTERED INDEX fkIdx_worker_id ON RestaurantEmployees (
  WorkerID ASC
 )


CREATE NONCLUSTERED INDEX fkIdx_manager_id ON RestaurantEmployees (
  ManagerID ASC
 )
```

Reports:

```
CREATE TABLE Reports (
 ReportID   int NOT NULL ,
 ManagerID  int NOT NULL ,
 CompanyID int NULL ,
 CustomerID int  NULL ,
 ReportDate nvarchar(50)  NULL ,


 CONSTRAINT PK_reports PRIMARY KEY CLUSTERED (ReportID ASC),
 CONSTRAINT FK_manager_id FOREIGN KEY (ManagerID)  REFERENCES Manager(ManagerID),
 CONSTRAINT FK_company_id_reports FOREIGN KEY (CompanyID)  REFERENCES
Company(CompanyID),
 CONSTRAINT FK_customer_id FOREIGN KEY (CustomerID)  REFERENCES
IndividualCustomer(CustomerID)
)
```

Tabela Reports przechowuje raporty, konstruowane przez managera firmy gastronomicznej, a więc kluczem obcym jest ManagerID. Inne kolumny jakie znajdziemy w tej tabeli to: ReportID, a więc ID odpowiedniego raportu, ID klientów dla których raporty są wystawiane (CustomerID, CompanyID) oraz data raportu (ReportDate). CompanyID oraz CustomerID są również kluczami obcymi.

Indeksy:

```
CREATE NONCLUSTERED INDEX fkIdx_manager_id  ON Reports (
  ManagerID ASC
 )
```

```
CREATE NONCLUSTERED INDEX fkIdx_company_id_reports ON Reports (
  CompanyID ASC
 )


CREATE NONCLUSTERED INDEX fkIdx_customer_id ON Reports (
  CustomerID ASC
 )
```

Invoices:

```
CREATE TABLE Invoices (
 InvoiceID         int NOT NULL ,
 SingleInvoice     bit  NULL ,
 CompanyID        int NOT NULL ,
 ManagerID         int NOT NULL ,
 InvoiceDate       nvarchar(50) NULL ,
 CollectiveInvoice bit  NULL ,


 CONSTRAINT PK_invoices PRIMARY KEY CLUSTERED (InvoiceID ASC),
 CONSTRAINT FK_company_id_invoices FOREIGN KEY (CompanyID)  REFERENCES
Company(CompanyID),
 CONSTRAINT FK_manager_id FOREIGN KEY (ManagerID)  REFERENCES Manager(ManagerID)
)
```

Tabela Invoices przechowuje informacje na temat faktur dla klientów (firm) (CompanyID) , wystawianych przez odpowiedniego pracownika (ManagerID). Są to dwa klucze obce w opisywanej tabeli. Ponadto znajdziemy takie kolumny jak: ID faktury (InvoiceID), datę wystawienia faktury (InvoiceDate), oraz dwa rodzaje faktur: zbiorcza (CollectiveInvoice) lub pojedyncza (SingleInvoice), które są typu bit i mogą przyjmować wartość NULL.

Indeksy:

```
CREATE NONCLUSTERED INDEX fkIdx_company_id_invoices ON Invoices (
  CompanyID ASC
 )


CREATE NONCLUSTERED INDEX fkIdx_manager_id ON Invoices (
  ManagerID ASC
 )
```

CompanyEmployee:

```
CREATE TABLE CompanyEmployee (
 CompanyEmployeeID int IDENTITY (1, 1) NOT NULL ,
 FirstName        nvarchar(40) NOT NULL ,
 LastName          nvarchar(20) NOT NULL ,
```

```
 Phone            nvarchar(20) NOT NULL ,
 Email            nvarchar(50) NOT NULL ,
 CompanyID        int NOT NULL ,


 CONSTRAINT PK_Customer PRIMARY KEY CLUSTERED (CompanyEmployeeID ASC),
 CONSTRAINT AK1_Customer_CustomerName UNIQUE NONCLUSTERED (FirstName ASC),
 CONSTRAINT FK_company_id_company_em FOREIGN KEY (CompanyID)  REFERENCES
Company(CompanyID),
 CONSTRAINT CheckEmail_CE CHECK(Email LIKE '%_@_%._%')
  )
```

Tabela CompanyEmployee przechowuje informacje dotyczące klientów jako pojedynczych pracowników z odpowiedniej firmy. Tabela posiada takie pola jak: ID klienta (CompanyEmployeeID), imię oraz nazwisko (FirstName, LastName), numer telefonu (Phone), adres e-mail (Email) oraz ID firmy której jest pracownikiem (CompanyID). Żadne z wartości nie mogą przyjmować wartości NULL, ponieważ są konieczne do dokonania rezerwacji. Adres e-mail również musi być określonej postaci podanej w CHECKU.

Indeksy:

```
CREATE NONCLUSTERED INDEX fkIdx_company_id_company_em ON CompanyEmployee (

  CompanyID ASC

   )
```

5. Spis widoków.

Widok menu:

```
CREATE VIEW Menu_View AS
SELECT ItemName, ItemDescription, ItemPrice
FROM MenuItems
GO
```

Widok niezarezerwowanych stolików:

```
CREATE VIEW Unreserved_Tables AS
SELECT [Tables].TableID
FROM [Tables] INNER JOIN Reservation ON Reservation.TableID=[Tables].TableID
WHERE Reservation.TableID IS NULL
GO
```

Widok do jakiej firmy należy konkretny pracownik z CompanyEmployee:

```
CREATE VIEW To_which_company_the_employee_belongs AS
SELECT CompanyName, FirstName + ' ' + LastName as worker
```

```
FROM CompanyEmployee INNER JOIN Company ON Company.CompanyID=
CompanyEmployee.CompanyID
GO
```

Widok niezatwierdzonych zamówień:

```
CREATE VIEW Unapproved_Orders AS
SELECT OrderID, ApprovedOrderID
FROM Orders
WHERE ApprovedOrderID='FALSE'
GO
```

Widok zatwierdzonych zamówień:

```
CREATE VIEW Approved_Orders AS
SELECT OrderID, ApprovedOrderID
FROM Orders
WHERE ApprovedOrderID='TRUE'
GO
```

Widok anulowanych płatności:

```
CREATE VIEW Canceled_Payments AS
SELECT PaymentID
FROM Payments
WHERE PaymentCancelDate IS NOT NULL
GO
```

Widok klientów indywidualnych którzy zarezerwowali stolik:

```
CREATE VIEW Customers_who_booked_a_table AS
SELECT FirstName + ' ' + LastName as customer, TableNumber
FROM Reservation
INNER JOIN IndividualCustomer ON
IndividualCustomer.CustomerID=Reservation.CustomerID
INNER JOIN [Tables] ON [Tables].TableID=Reservation.TableID
GO
```

Widok firm które zarezerwowały stolik:

```
CREATE VIEW Company_who_booked_a_table AS
SELECT CompanyName, TableNumber
FROM Reservation INNER JOIN Company ON Company.CompanyID=Reservation.CompanyID
INNER JOIN [Tables] ON [Tables].TableID=Reservation.TableID
GO
```

Widok pojedynczego pracownika z konkretnej firmy, który zarezerwował stolik:

```
CREATE VIEW Company_Employee_who_booked_a_table AS
SELECT FirstName + ' ' + LastName as company_employee, TableNumber
```

```
FROM Reservation INNER JOIN CompanyEmployee ON
CompanyEmployee.CompanyEmployeeID=Reservation.CompanyEmployeeID
INNER JOIN [Tables] ON [Tables].TableID=Reservation.TableID
GO
```

Widok raportu dotyczącego rezerwacji za pomocą formularza internetowego przez klienta indywidualnego:

```
CREATE VIEW Table_Online_Reservation_Raport_for_Individual_Customer AS
SELECT ManagerID, ReportDate, FirstName + ' ' + LastName as customer, TableNumber,
ReservationDate
FROM Reports INNER JOIN IndividualCustomer ON
IndividualCustomer.CustomerID=Reports.CustomerID
INNER JOIN Reservation ON Reservation.CustomerID=IndividualCustomer.CustomerID
INNER JOIN [Tables] ON [Tables].TableID=Reservation.TableID
WHERE OnilneReservation IS NOT NULL
GO
```

-

Widok raportu dotyczącego rezerwacji telefonicznej przez klienta indywidualnego:

```
CREATE VIEW Table_Phone_Reservation_Raport_for_Individual_Customer AS
SELECT ManagerID, ReportDate, FirstName + ' ' + LastName as customer, TableNumber,
ReservationDate
FROM Reports INNER JOIN IndividualCustomer ON
IndividualCustomer.CustomerID=Reports.CustomerID
INNER JOIN Reservation ON Reservation.CustomerID=IndividualCustomer.CustomerID
INNER JOIN [Tables] ON [Tables].TableID=Reservation.TableID
WHERE ReservationByPhone IS NOT NULL
GO
```

Widok raportu dotyczącego rezerwacji za pomocą formularza internetowego przez firmę:

```
CREATE VIEW Table_Online_Reservation_Raport_for_Company AS
SELECT ManagerID, ReportDate, CompanyName, TableNumber, ReservationDate
FROM Reports INNER JOIN Company ON Company.CompanyID=Reports.CompanyID
INNER JOIN Reservation ON Reservation.CompanyID=Company.CompanyID
INNER JOIN [Tables] ON [Tables].TableID=Reservation.TableID
WHERE OnilneReservation IS NOT NULL
GO
```

Widok raportu dotyczącego rezerwacji telefonicznej przez firmę:

```
CREATE VIEW Table_Phone_Reservation_Raport_for_Company AS
SELECT ManagerID, ReportDate, CompanyName, TableNumber, ReservationDate
FROM Reports INNER JOIN Company ON Company.CompanyID=Reports.CompanyID
INNER JOIN Reservation ON Reservation.CompanyID=Company.CompanyID
INNER JOIN [Tables] ON [Tables].TableID=Reservation.TableID
WHERE ReservationByPhone IS NOT NULL
GO
```

Widok raportu dotyczącego zniżek:

```
CREATE VIEW Discount_Raport_for_Individual_Customer AS
SELECT ManagerID, ReportDate, FirstName + ' ' + LastName as customer, ItemName,
UnitPrice, Discount
FROM Reports INNER JOIN IndividualCustomer ON
IndividualCustomer.CustomerID=Reports.CustomerID
INNER JOIN Orders ON Orders.CustomerID=IndividualCustomer.CustomerID
INNER JOIN OrderDetails ON OrderDetails.OrderID=Orders.OrderID
INNER JOIN MenuItems ON MenuItems.ItemID=OrderDetails.ItemID
WHERE Discount IS NOT NULL
GO
```

6. Spis procedur.

Procedura dodająca dane do tabeli [Tables]:

Argumentami jest ID stolika (`@TableID`), liczba miejsc (`@NumberOfSeats`) przy stoliku oraz numer stolika (`@TableNumber`).

```
CREATE PROCEDURE dbo.Add_Table  @TableID int , @NumberOfSeats int , @TableNumber int
AS
BEGIN
     SET NOCOUNT ON

     INSERT INTO dbo.[Tables]
          (
            TableID,
            NumberOfSeats,
            TableNumber
          )
     VALUES
          (
             @TableID,
            @NumberOfSeats,
            @TableNumber
          )

  END
GO
```

Procedura dodająca dane do tabeli Company:

Argumentami jest ID firmy (`@CompanyID`), nazwa firmy (`@CompanyName`), numer telefonu (`@CompanyPhone`), adres e-mail firmy (`@CompanyEmail`), adres siedziby firmy (`@CompanyAddress`).

```
CREATE PROCEDURE dbo.Add_Company  @CompanyID int , @CompanyName nvarchar(50) ,
@CompanyPhone  nvarchar(50), @CompanyEmail nvarchar(50), @CompanyAddress
nvarchar(50)  AS
BEGIN
     SET NOCOUNT ON
      SET IDENTITY_INSERT dbo.Company ON
     INSERT INTO dbo.Company
          (
```

```
            CompanyID,
                  CompanyName,
                  CompanyPhone,
                  CompanyEmail,
                  CompanyAddress
            )
      VALUES
            (
            @CompanyID,
                  @CompanyName,
                  @CompanyPhone,
                  @CompanyEmail,
                  @CompanyAddress
            )
      SET IDENTITY_INSERT dbo.Company OFF

  END
GO
```

Procedura dodająca dane do tabeli IndividualCustomer:

Argumentami jest ID klienta (`@CustomerID`), imię klienta (`@FirstName`), nazwisko klienta (`@LastName`), numer telefonu klienta (`@Phone_no`), adres zamieszkania klienta (`@Address`) oraz adres e-mail klienta (`@Email`).

```
CREATE PROCEDURE dbo.Add_Individual_Customer  @CustomerID int , @FirstName
nvarchar(50) , @LastName  nvarchar(50), @Phone_no nvarchar(50), @Address
nvarchar(50), @Email nvarchar(50) AS
BEGIN
      SET NOCOUNT ON
       SET IDENTITY_INSERT dbo.IndividualCustomer ON
      INSERT INTO dbo.IndividualCustomer
            (
            CustomerID,
                  FirstName,
                  LastName,
                  Phone_no,
                  [Address],
                   Email
            )
      VALUES
            (
            @CustomerID,
                  @FirstName,
                  @LastName,
                  @Phone_no,
                  @Address,
                  @Email
            )
      SET IDENTITY_INSERT dbo.IndividualCustomer OFF

  END
GO
```

Procedura dodająca dane do tabeli RestaurantWorker:

Argumentem jest ID pracownika (@WorkerID).

```
CREATE PROCEDURE dbo.Add_Restaurant_Worker  @WorkerID int  AS
BEGIN
     SET NOCOUNT ON
      SET IDENTITY_INSERT dbo.RestaurantWorker ON
     INSERT INTO dbo.RestaurantWorker
          (
            WorkerID

          )
     VALUES
          (
            @WorkerID

          )
     SET IDENTITY_INSERT dbo.RestaurantWorker OFF

  END
GO
```

Procedura dodająca dane do tabeli Reservation:

Argumentami są: ID rezerwacji (@ReservationID), data rezerwacji (@ReservationDate), imię klienta indywidualnego (@FirstNameIC), nazwisko klienta indywidualnego (@LastNameIC), nazwa firmy (@CompanyName), ID pracownika (@Worker), które wybierane jest z tabeli RestaurantWorker, numer stolika (@TableNumber), imię klienta z wybranej firmy (@FirstNameCE), nazwisko klienta z wybranej firmy (@LastNameCE).
Niektóre z wartości mogą być NULL-ami, ponieważ możemy dodać do tabeli Reservation, rezerwację wybranego klienta (indywidualnego, firmy lub pracownika firmy) wpisując dane w wybranego dla tego klienta pola, a uzupełniając inne NULL-ami.
CustomerID wybierany jest za za pomocą argumentów @FirstNameIC oraz @LastNameIC, poprzez wyszukanie odpowiedniego ID w tabeli IndividualCustomer. Na tej samej zasadzie działa wybieranie ID pracownika firmy, oraz firmy (w przypadku firmy wyszukiwanie następuje poprzez nazwę firmy).

```
CREATE PROCEDURE dbo.Add_Reservation
@ReservationID int ,
@ReservationDate nvarchar(50) ,
@FirstNameIC nvarchar(50) =NULL,
@LastNameIC nvarchar(50)=NULL,
@CompanyName nvarchar(50)=NULL,
@Worker  int=NULL,
@TableNumber int,
@FirstNameCE  nvarchar(50) = NULL,
@LastNameCE nvarchar(50) =NULL
AS
BEGIN
     SET NOCOUNT ON
      DECLARE @CustomerID AS int
            SET @CustomerID = (
                  SELECT CustomerID
                  FROM IndividualCustomer
```

```
                WHERE FirstName = @FirstNameIC AND LastName=@LastNameIC

        )
    DECLARE @CompanyID AS int
        SET @CompanyID = (
            SELECT CompanyID
            FROM Company
            WHERE CompanyName=@CompanyName
        )
    DECLARE @WorkerID AS int
        SET @WorkerID = (
            SELECT WorkerID
            FROM RestaurantWorker
            WHERE WorkerID=@Worker
        )
    DECLARE @TableID AS int
        SET @TableID = (
            SELECT TableID
            FROM [Tables]
            WHERE TableNumber=@TableNumber
        )
    DECLARE @CompanyEmplID AS int
        SET @CompanyEmplID  = (
            SELECT CompanyEmployeeID
            FROM CompanyEmployee
            WHERE FirstName=@FirstNameCE AND LastName=@LastNameCE
        )
    DECLARE @ReservationByPhone AS nvarchar(50) = NULL
    IF @CompanyID IS NULL AND @CustomerID IS NULL AND @CompanyEmplID IS NOT NULL
    BEGIN
        SET @ReservationByPhone  = (
            SELECT Phone
            FROM CompanyEmployee
            WHERE CompanyEmployeeID=@CompanyEmplID
        )
    END
    ELSE IF @CompanyID IS NULL AND @CustomerID IS NOT NULL AND @CompanyEmplID IS
NULL
    BEGIN
        SET @ReservationByPhone  = (
            SELECT Phone_no
            FROM IndividualCustomer
            WHERE CustomerID=@CustomerID

        )
    END
    ELSE IF @CompanyID IS NOT NULL AND @CustomerID IS  NULL AND @CompanyEmplID IS
NULL
    BEGIN
        SET  @ReservationByPhone  = (
            SELECT CompanyPhone
            FROM Company
            WHERE CompanyID=@CompanyID
        )
    END
    DECLARE @OnlineResrvationName AS nvarchar(50) = NULL
    IF @CompanyID IS NULL AND @CustomerID IS NULL AND @CompanyEmplID IS NOT NULL
    BEGIN
        SET @OnlineResrvationName  = (
            SELECT FirstName+ ' ' + LastName
            FROM CompanyEmployee
            WHERE FirstName=@FirstNameCE AND LastName=@LastNameCE
```

```
            )
      END
      ELSE IF @CompanyID IS NULL AND @CustomerID IS NOT NULL AND @CompanyEmplID IS
NULL
      BEGIN
            SET @OnlineResrvationName  = (
                  SELECT FirstName+ ' ' + LastName
                  FROM IndividualCustomer
                  WHERE FirstName = @FirstNameIC AND LastName=@LastNameIC
            )
      END
      ELSE IF @CompanyID IS NOT NULL AND @CustomerID IS  NULL AND @CompanyEmplID IS
NULL
      BEGIN
            SET @OnlineResrvationName  = (
                  SELECT CompanyName
                  FROM Company
                  WHERE CompanyName=@CompanyName
            )
      END
      INSERT INTO dbo.Reservation
            (
             ReservationID,
            ReservationByPhone,
            ReservationDate,
            CompanyID,
            CustomerID,
            WorkerID,
            OnilneReservation,
            TableID,
            CompanyEmployeeID
            )
      VALUES
            (
             @ReservationID,
            @ReservationByPhone,
            @ReservationDate,
            @CompanyID,
            @CustomerID,
            @WorkerID,
            @OnlineResrvationName,
            @TableID,
            @CompanyEmplID
            )
END
  GO
```

Procedura dodająca dane do tabeli Menu:

Argumentami są ID menu (`@MenuID`) oraz wersja menu (`@MenuVersion`) w postaci daty.

```
CREATE PROCEDURE dbo.Add_Menu @MenuID int , @MenuVersion nvarchar(50)  AS
BEGIN
      SET NOCOUNT ON
      INSERT INTO dbo.Menu
            (
             MenuID,
            MenuVersion

            )
```

```
      VALUES
            (
             @MenuID,
            @MenuVersion
            )

END
```

Procedura dodająca dane do tabeli SeaFoodDishes:

Argumentami są ID owoców morza (@SeaFoodID), dzień (Czwartek, Piątek lub Sobota) (@PossibleDaysToOrder), oraz określenie czy składniki są na wyczerpaniu w postaci booleana (true lub false) (@DepletionOfIntermediates).

```
CREATE PROCEDURE dbo.Add_SeaFoodDishes  @SeaFoodID int , @PossibleDaysToOrder
nvarchar(50), @DepletionOfIntermediates bit  AS
BEGIN
      SET NOCOUNT ON

      INSERT INTO dbo.SeaFoodDishes
            (
             SeaFoodID,
            PossibleDaysToOrder,
            DepletionOfIntermediates

            )
      VALUES
            (
             @SeaFoodID,
            @PossibleDaysToOrder,
            @DepletionOfIntermediates
            )


  END
GO
```

Procedura dodająca dane do tabeli MenuItems:

Argumentami są ID produktu (@ItemID), nazwa produktu (@ItemName), opis produktu (@ItemDescription), cena produktu (@ItemPrice), wersja menu (@MenuVersion), i ewentualne id owocow morza (@SeaFoodID), które może być Nullem, ponieważ nie jest dostępne zawsze w menu (tylko w określone dni: patrz wyżej).

```
CREATE PROCEDURE dbo.Add_Menu_Items @ItemID int , @ItemName nvarchar(50),
@ItemDescription nvarchar(100) , @ItemPrice money, @MenuVersion nvarchar(100),
@SeaFoodID   int= NULL AS
BEGIN
      SET NOCOUNT ON
       DECLARE @MenuID AS int
            SET @MenuID  = (
                  SELECT MenuID
                  FROM Menu
                  WHERE MenuVersion=@MenuVersion
            )

      INSERT INTO dbo.MenuItems
            (
```

```
            ItemID,
            ItemName,
            ItemDescription,
            ItemPrice,
            MenuID,
            SeaFoodID
          )
    VALUES
          (
            @ItemID,
            @ItemName,
            @ItemDescription,
            @ItemPrice,
            @MenuID,
            @SeaFoodID )
  END
GO
```

Procedura dodająca dane do tabeli Payments:

Argumentami są id płatności (@PaymentID), oraz odpowiednie rodzaje płatności (gotówka (@CashPayment), przelew (@BankTransferPayment), płatność kartą(@CardPayment)) oraz data płatności (@PaymentDate). Rodzaj płatności zostaje dodany poprzez ustawienie wartości na true. Ważne jest to, że podane trzy rodzaje płatności nie mogą przyjmować wartości (false, false, false) lub (true, true, true) lub nie może być sytuacji w której dwa z rodzajów płatności mają wartość true. Oznacza to zatem, że tylko jeden rodzaj płatności może być ustawiony na true (ponieważ płatności w tym samym momencie można dokonywać tylko na jeden sposób).
Zatem poprawnym dodaniem wartości do określonych sposobów płatności byłoby: (true, false, false) lub (false, true, false) lub (false, false, true).

```
CREATE PROCEDURE dbo.Add_Payments  @PaymentID int , @CashPayment nvarchar(50),
@BankTransferPayment bit, @CardPayment bit,  @PaymentDate  nvarchar(50) AS
BEGIN
      SET NOCOUNT ON

      INSERT INTO dbo.Payments
            (
              PaymentID,
            CashPayment,
            BankTransferPayment,
            CardPayment,
            PaymentDate

            )
      VALUES
            (
              @PaymentID,
            @CashPayment,
            @BankTransferPayment,
            @CardPayment,
            @PaymentDate
            )


  END
GO
```

Procedura dodająca dane do tabeli Orders:

Argumentami są id zamówienia (@OrderID), imię klietna konkretnej firmy (@FirstNameCE), nazwisko klienta konkretnej firmy (@LastNameCE), data zamówienia (@OrderDate), imię klienta indywidualnego (@FirstNameIC), nazwisko klienta indywidualnego (@LastNameIC), nazwa firmy (@CompanyName), status zamówienia (@OrderStatus), płatność (@Payment), stwierdzenie czy zamowienia jest zatwierdzone (true/false) (@ApprovedOrderID), rodzaj zamówienia(online/phone) (@OrderTypeName), ID pracownika obsługującego zamówienie (@Worker).
ID pracownika wybierane jest z tabeli RestaurantWorker.
Wartości dla konkretnych klientów są NULL-ami, więc dane dodajemy wybierając klienta którego zamówienie chcemy dodać, poprzez uzupełnienie odpowiednich pól dla klienta (dla klienta indywidualnego oraz pracownika firmy uzupełniamy imię oraz nazwisko, dla firmy –nazwę firmy), a pola dla pozostałych klientów ustawiamy na NULL.
Status zamówienia może być odpowiednio: ‘completed’ lub ‘accepted’.

```
CREATE PROCEDURE dbo.Add_Orders
@OrderID  int ,
@FirstNameCE nvarchar(50)=NULL,
@LastNameCE nvarchar(50)=NULL,
@OrderDate  nvarchar(50),
@FirstNameIC nvarchar(50)=NULL,
@LastNameIC nvarchar(50)=NULL,
@CompanyName nvarchar(50)=NULL,
@OrderStatus nvarchar(50),
@Payment int,
@ApprovedOrderID  bit,
@OrderTypeName nvarchar(50),
@Worker int AS
BEGIN
     SET NOCOUNT ON
      SET IDENTITY_INSERT dbo.Orders ON
     DECLARE @CustomerID AS int
            SET @CustomerID = (
                   SELECT CustomerID
                   FROM IndividualCustomer
                   WHERE FirstName = @FirstNameIC AND LastName=@LastNameIC

            )
     DECLARE @CompanyID AS int
            SET @CompanyID = (
                   SELECT CompanyID
                   FROM Company
                   WHERE CompanyName=@CompanyName
            )
     DECLARE @WorkerID AS int
            SET @WorkerID = (
                   SELECT WorkerID
                   FROM RestaurantWorker
                   WHERE WorkerID=@Worker
            )
     DECLARE @PaymentID AS int
            SET @PaymentID = (
                   SELECT PaymentID
                   FROM Payments
```

```
                WHERE PaymentID=@Payment
        )
    DECLARE @CompanyEmplID AS int
        SET @CompanyEmplID  = (
                SELECT CompanyEmployeeID
                FROM CompanyEmployee
                WHERE FirstName=@FirstNameCE AND LastName=@LastNameCE
        )
    INSERT INTO dbo.Orders
        (
         OrderID,
        CompanyEmployeeID,
        OrderDate,
        CustomerID,
        CompanyID,
        OrderStatus,
        PaymentID,
        ApprovedOrderID,
        OrderTypeName,
        WorkerID
        )
    VALUES
        (
          @OrderID,
        @CompanyEmplID,
        @OrderDate,
        @CustomerID,
        @CompanyID,
        @OrderStatus,
        @PaymentID,
        @ApprovedOrderID,
        @OrderTypeName,
        @WorkerID
        )
    SET IDENTITY_INSERT dbo.Orders OFF

  END
GO
```

Procedura dodająca dane do tabeli OrderDetails:

Argumentami są ID zamówienia (`@OrderID`), nazwa produktu (`@ItemName`) wybierana z tabeli MenuItems, ilość produktów (`@Amount`), ilość zamówień (`@AmountOfOrders`), liczba dni trwania zamówienia (`@NumberOfDaysOfPlacedOrders`). Ta ostatnia może być NULL-em, ponieważ wykorzystywana jest do obliczenia zniżki, która z kolei obliczana jest w różnorodny sposób: albo ze względu na liczbę zamówień albo ze względu na liczbę dni przez ile dane zamówienie było składane. Dodatkowo do obliczenia rabatów wykorzystywane są dwie funkcje (w zależności, które z rodzajów zniżek chcemy obliczyć): `dbo.Calculate_Discount_For_The_Number_Of_Orders` oraz `dbo.Calculate_Discount_For_The_Number_Of_Days`.

```
CREATE PROCEDURE dbo.Add_Order_Details
@OrderID  int ,
@ItemName nvarchar(50),
@Amount int ,
@AmountOfOrders int,
@NumberOfDaysOfPlacedOrders int = NULL
AS
BEGIN
```

```
      SET NOCOUNT ON
       DECLARE @ItemID AS int
            SET @ItemID = (
                  SELECT ItemID
                  FROM MenuItems
                  WHERE ItemName=@ItemName
            )
      DECLARE @UnitPrice AS decimal(5,2)
            SET @UnitPrice = @Amount *(
                  SELECT ItemPrice
                  FROM MenuItems
                  WHERE ItemName=@ItemName
            )
            DECLARE @Discount AS decimal(5,2)
            IF (@NumberOfDaysOfPlacedOrders IS NULL)
            BEGIN
            SET @Discount = (select
  dbo.Calculate_Discount_For_The_Number_Of_Orders(@AmountOfOrders,
  @Amount*@UnitPrice))
            END
            ELSE IF (@NumberOfDaysOfPlacedOrders IS NOT NULL)
            BEGIN
            SET @Discount =(select
  dbo.Calculate_Discount_For_The_Number_Of_Days( @Amount*@UnitPrice,
  @NumberOfDaysOfPlacedOrders))
            END

      INSERT INTO dbo.OrderDetails
           (
              OrderID,
            ItemID,
            Amount,
            UnitPrice,
            AmountOfOrders,
            Discount
           )
      VALUES
           (
              @OrderID,
            @ItemID,
            @Amount,
            @UnitPrice,
            @AmountOfOrders,
            @Discount
           )


  END
GO
```

Procedura dodająca dane do tabeli Manager:

Argumentem jest ID managera (@ManagerID).

```
CREATE PROCEDURE dbo.Add_Manager  @ManagerID int AS
BEGIN
      SET NOCOUNT ON
       SET IDENTITY_INSERT dbo.Manager ON
      INSERT INTO dbo.Manager
           (
```

```
            ManagerID

        )
    VALUES
        (
            @ManagerID
        )
    SET IDENTITY_INSERT dbo.Manager OFF

  END
GO
```

Procedura dodająca dane do tabeli RestaurantEmployee:

Argumentem są ID pracownika (`@Worker`) wybierane z tabeli RestaurantWorker, ID managera (`@Manager`) wybierane z tabeli Manager, imię (`@FirstName`), nazwisko (`@LastName`), data zatrudnienia (`@HireDate`), adres zamieszkania (`@Address`), numer telefonu (`@HomePhone`).

```
CREATE PROCEDURE dbo.Add_Restaurant_Employee
@Worker int,
@Manager int,
@FirstName nvarchar(50),
@LastName  nvarchar(50),
@HireDate  nvarchar(50)=NULL,
@Address  nvarchar(50),
@HomePhone nvarchar(50)=NULL  AS
BEGIN
    SET NOCOUNT ON
       DECLARE @WorkerID AS int
            SET @WorkerID = (
                SELECT WorkerID
                FROM RestaurantWorker
                WHERE WorkerID=@Worker
            )
      DECLARE @ManagerID AS int
            SET @ManagerID = (
                SELECT ManagerID
                FROM Manager
                WHERE ManagerID=@Manager
            )
    INSERT INTO dbo.RestaurantEmployees
        (
          WorkerID,
          ManagerID,
          FirstName,
          LastName,
          HireDate,
          [Address],
          HomePhone

        )
    VALUES
        (
          @WorkerID,
          @ManagerID,
          @FirstName,
```

```
            @LastName,
            @HireDate,
            @Address,
            @HomePhone
        )


END
GO
```

Procedura dodająca dane do tabeli Reports:

Argumentami są ID raportu (@ReportID), imię managera piszącego raport (@ManagerName), nazwisko managera piszącego raport (@ManagerSurname), nazwa firmy (@CompanyName), imię klienta indywidualnego (@FirstNameIC), nazwisko klienta indywidualnego (@LastNameIC), data raportu (@ReportDate). Wartości dla firmy i klienta są ustawione na NULL, więc dodając dane do tabeli wybieramy konkretnego klienta, dla którego raport ma zostać sporządzony, a pozostałego klienta ustawiamy na NULL (nie możemy w jednym momencie sporządzić raportu dla wszystkich klientów).

```
CREATE PROCEDURE dbo.Add_Report
@ReportID int,
@ManagerName nvarchar(50),
@ManagerSurname nvarchar(50),
@CompanyName nvarchar(50)=NULL,
@FirstNameIC  nvarchar(50)=NULL,
@LastNameIC nvarchar(50)=NULL,
@ReportDate   nvarchar(50) AS
BEGIN
     SET NOCOUNT ON
      DECLARE @CustomerID AS int
            SET @CustomerID = (
                  SELECT CustomerID
                  FROM IndividualCustomer
                  WHERE FirstName = @FirstNameIC AND LastName=@LastNameIC

            )
     DECLARE @CompanyID AS int
            SET @CompanyID = (
                  SELECT CompanyID
                  FROM Company
                  WHERE CompanyName=@CompanyName
            )
     DECLARE @ManagerID AS int
            SET @ManagerID = (
                  SELECT ManagerID
                  FROM RestaurantEmployees
                  WHERE FirstName = @ManagerName AND LastName=@ManagerSurname

            )
      INSERT INTO dbo.Reports
           (
              ReportID,
             ManagerID,
             CompanyID,
             CustomerID,
             ReportDate
```

```
        )
    VALUES
        (
          @ReportID,
         @ManagerID,
         @CompanyID,
         @CustomerID,
         @ReportDate )


  END
GO
```

Procedura dodająca dane do tabeli Invoices:

Argumentami są ID faktury (@InvoiceID), określenie rodzaju faktury (zbiorczej (@CollectiveInvoice) lub pojedynczej (@Singleinvoice)) ustawiając jedną z możliwości true a inną na false, nazwa firmy (@CompanyName), imię managera wystawiającego fakturę (@ManagerName), nazwisko managera wystawiającego fakturę (@ManagerSurname), data wystawienia faktury (@InvoiceDate).

```
CREATE PROCEDURE dbo.Add_Invoice
@InvoiceID int,
@Singleinvoice bit = NULL,
@CompanyName nvarchar(50)=NULL,
@ManagerName nvarchar(50),
@ManagerSurname nvarchar(50),
@InvoiceDate  nvarchar(50)= NULL ,
@CollectiveInvoice bit = NULL
AS
BEGIN
     SET NOCOUNT ON
    DECLARE @CompanyID AS int
           SET @CompanyID = (
                  SELECT CompanyID
                  FROM Company
                  WHERE CompanyName=@CompanyName
           )
    DECLARE @ManagerID AS int
           SET @ManagerID = (
                  SELECT ManagerID
                  FROM RestaurantEmployees
                  WHERE FirstName = @ManagerName AND LastName=@ManagerSurname

           )
     INSERT INTO dbo.Invoices
          (
            InvoiceID,
                 SingleInvoice,
                 CompanyID,
                 ManagerID,
                 InvoiceDate,
                 CollectiveInvoice
```

```
      )
VALUES
      (
         @InvoiceID,
               @SingleInvoice,
               @CompanyID,
               @ManagerID,
               @InvoiceDate,
               @CollectiveInvoice )


END
GO
```

Procedura dodająca dane do tabeli CompanyEmployee:

Argumentami są ID pracownika firmy (`@CompanyEmployeeID`), imię (`@FirstName`), nazwisko (`@LastName`), numer telefonu (`@Phone`), adres e-mail (`@Email`), nazwa firmy (`@CompanyName`).

```
CREATE PROCEDURE dbo.Add_CompanyEmployee
@CompanyEmployeeID int ,
@FirstName        nvarchar(40),
@LastName          nvarchar(20),
@Phone            nvarchar(20),
@Email            nvarchar(50) ,
@CompanyName      nvarchar(20)
AS
BEGIN
     SET NOCOUNT ON
       SET IDENTITY_INSERT dbo.CompanyEmployee ON
     DECLARE @CompanyID AS int
           SET @CompanyID = (
                  SELECT CompanyID
                  FROM Company
                  WHERE CompanyName=@CompanyName
           )
     INSERT INTO dbo.CompanyEmployee
          (
             CompanyEmployeeID,
             FirstName,
             LastName,
             Phone,
             Email,
             CompanyID

          )
     VALUES
          (
             @CompanyEmployeeID,
             @FirstName,
             @LastName,
             @Phone,
             @Email,
             @CompanyID )
```

```
   SET IDENTITY_INSERT dbo.CompanyEmployee OFF
  END
GO
```

Procedura anulująca stolik:

Argumentem jest numer stolika (`@TableNumber`), który chcemy anulować.

```
CREATE PROCEDURE dbo. Cancel_Table
@TableNumber int
AS
BEGIN
     SET NOCOUNT ON
     DECLARE @TableID AS int
            SET @TableID = (
                   SELECT TableID
                   FROM [Tables]
                   WHERE TableNumber=@TableNumber
            )
     UPDATE Reservation
     SET CancelDate = GETDATE()
     WHERE TableID=@TableID
     END
GO
```

Procedura anulująca płatność:

Argumentem jest ID płatności (`@PaymentID`), którą chcemy anulować.

```
CREATE PROCEDURE dbo. Cancel_Payment
@PaymentID int
AS
BEGIN
     SET NOCOUNT ON
     UPDATE Payments
     SET PaymentCancelDate = GETDATE()
     WHERE PaymentID=@PaymentID
     END
   GO
```

7. Spis funkcji.

Funkcja obliczająca zniżkę ze względu na liczbę zamówień:

Argumentami jest ilość zamówień (`@Amount`) oraz cena (`@price`) za pojedyncze zamówienie.

```
CREATE FUNCTION dbo.Calculate_Discount_For_The_Number_Of_Orders (@Amount int, @price
money)
RETURNS decimal(5,2)
AS BEGIN
DECLARE @AmountOfDiscount decimal(5,2)
IF (@Amount > 10)
```

```
BEGIN
	SET @AmountOfDiscount = (@Amount * @price)/100
END
ELSE
BEGIN
SET @AmountOfDiscount =NULL
END
    RETURN @AmountOfDiscount
END
GO
```

Funkcja obliczająca zniżkę ze względu na liczbę dni zamówienia(zniżka jednorazowa):

Argumentami jest cena za pojedyncze zamówienie (@price) oraz liczba dni przez ile zamówienie było zamawiane (@numDays). Jeżeli dane zamówienie było składane przez siedem dni, przysługuje jednorazowa zniżka.

```
CREATE FUNCTION dbo.Calculate_Discount_For_The_Number_Of_Days  (@price money, @numDays
int)
RETURNS decimal(5,2)
AS BEGIN
DECLARE @AmountOfDiscount decimal(5,2)
IF (@numDays=7)
BEGIN
	SET @AmountOfDiscount = 5.00
END
ELSE
BEGIN
SET @AmountOfDiscount = NULL
END
    RETURN @AmountOfDiscount
END
   GO
```

8. Spis triggerów.

Ustawianie limitu liczby stolików:

Maksymalna liczba stolików nie może być większa niż 25. Zatem jeżeli użytkownik podczas wykonywania INSERT doda rekord w momencie, gdy ich liczba wynosi już 25 – transakcja zostanie cofnięta.

```
CREATE TRIGGER LimitTable
ON [Tables]
AFTER INSERT
AS
    DECLARE @tableCount int
    SELECT @tableCount = COUNT(*)
```

```
    FROM [Tables]

    IF @tableCount > 25
    BEGIN
        ROLLBACK
    END
GO
```

Usuwanie zamówienia z anulowaną płatnością:

```
CREATE TRIGGER Cancel_Record
ON Payments
FOR DELETE
AS
DELETE FROM Orders
       WHERE PaymentID =
       (SELECT PaymentID FROM Payments WHERE PaymentCancelDate IS NOT NULL)
GO
```

Usuwanie rezerwacji:

```
CREATE TRIGGER Cancel_Reservation_trigg
ON Reservation
FOR DELETE
AS
DELETE FROM Reservation
       WHERE CancelDate IS NOT NULL
GO
```

9. Wygenerowane dane.

Dane zostały wygenerowane za pomocą generatora online. Znajdują się w katalogu: wygenerowane_dane. Są tam dane dla każdej tabeli. Odpowiednia nazwa pliku to odpowiednia tabela. Zatem łącznie jest 16 plików (16 tabel).

10. Wyszczególnienie najważniejszych informacji.

- **Tables** – stoliki mają swój unikatowy numer, minimalna liczba miejsc przy stoliku to dwa. Ponadto w restauracji może znajdować się maksymalnie 25 stolików.

- **Reservation** – rezerwacje stolików mogą odbywać się na dwa sposoby: telefonicznie lub za pomocą formularza online (jedna z wybranych opcji). Rezerwacji mogą dokonywać klienci indywidualni, firmy oraz wybrani klienci z określonych firm, z tym że Ci ostatni dokonują tego imiennie (a nie na nazwę firmy). Rezerwację można anulować.

- **Menu, MenuItems** – każde menu posiada datę swojej modyfikacji, która dokonywana jest co najmniej raz na dwa tygodnie. W menu co jakiś czas mogą pojawiać się owoce morza (czwartki, piątki, soboty).

- **OrderDetails, Orders** – zamówienia realizowane są przez pracownika restauracji (RestaurantWorker). Możliwe jest anulowanie zamówienia poprzez anulowanie płatności. Status zamówienia może być ‘completed’ jeżeli dokonano płatności lub ‘accepted’ jeżeli zamówienie zostało przyjęte ale jeszcze nieopłacone.

- **RestaurantWorker, Manager** – pracownicy, gdzie do każdego należą inne obowiązki. RestaurantWorker ma za zadanie realizować zamówienia, płatności oraz rezerwacje stolików. Z kolei manager spisuje raporty oraz wystawia faktury dla poszczególnych klientów.

- **IndividualCustomer, Company, CompanyEmployee** – klienci którzy mogą zamawiać produkty oraz rezerwować stoliki. Dla wszystkich klientów wystawiane są raporty, natomiast faktura wystawiana jest tylko dla firmy (może być pojedyncza lub zbiorcza).

- **Payments** – są trzy sposoby płatności (kartą, przelewem lub gotówką). Płatność może zostać anulowana