# OPIS PROCEDUR I FUNKCJI ORAZ KOD JE GENERUJĄCY

Przypomnienie ról systemowych:

* Chef - odczyt OrderDetails
* Accountant - generowanie faktur i raportów, przeglądanie Orders, kursy walut
* Waiter - dostep do Orders i OrdersDetails
* Menager - nadzoruje całość, kontrola Employees
* Administrator IT - kontrola nad całym systemem
* WarehouseMenager - nadzoruje zamówienia i dba o uzupełnianie warehouse

Każda procedura zawiera:

1. krótki opis
2. użytkownika, który może go wywołać / zobaczyć.
3. argumenty, które przyjmuje
4. wyjście lub stwierdzenie, że go nie ma.
5. Kod procedury.

### Podsumowanie zamówienia jako lista produktów (OrderID)

* Umożliwia otrzymanie informacji z czego składało się dane zamówienie
* Może go wyświetlić Waiter, Accountant oraz Chef.
* Przyjmuje numer zamówienia
* Zwraca listę składników zamówienia

CREATE PROCEDURE *uspListOrderDetails*(  
 @orderID AS INT  
)  
AS  
BEGIN  
 IF @orderID is NULL or @orderID not in (SELECT OrderID from Orders)  
 BEGIN  
 print N'invalid orderID';  
 END  
 ELSE  
 BEGIN  
 SELECT W.Name, OD.Quantity as Quan, OD.UnitPrice as Price, O.CurrencyName  
 FROM OrderDetails OD  
 INNER JOIN Orders O on O.OrderID = OD.OrderID  
 INNER JOIN Warehouse W on W.MealID = OD.MealID  
 WHERE O.OrderID = @orderID  
 END  
END  
GO;  
  
--przykładwe odpalenie  
EXECUTE uspListOrderDetails @orderID = 7;

### Podsumowanie zamówienia jako sumaryczna cena (OrderID)

* Umożliwia otrzymanie sumarycznego kosztu zamówienia
* Może go wyświetlić Waiter, Accountant oraz Chef.
* Przyjmuje numer zamówienia
* Zwraca koszt zamówienia

CREATE PROCEDURE *uspOrderCost*(  
 @orderID AS INT  
)  
AS  
BEGIN  
 if (select DiscountID from Orders where Orders.OrderID = @orderID) is NULL  
 BEGIN  
 SELECT SUM(Quantity \* W.UnitPrice) as sum, WhenOrdered as 'Order date'  
 FROM OrderDetails OD  
 INNER JOIN Orders O on O.OrderID = OD.OrderID  
 INNER JOIN Warehouse W on W.MealID = OD.MealID  
 WHERE O.OrderID = @orderID  
 GROUP BY WhenOrdered  
 END  
 ELSE  
 BEGIN  
 SELECT SUM(Quantity \* W.UnitPrice \* (1 - D.DiscountValue)) as sum, WhenOrdered as 'Order date'  
 FROM OrderDetails OD  
 INNER JOIN Orders O on O.OrderID = OD.OrderID  
 INNER JOIN Warehouse W on W.MealID = OD.MealID  
 INNER JOIN Discounts D on D.DiscountID = O.DiscountID  
 WHERE O.OrderID = @orderID  
 GROUP BY WhenOrdered  
 END  
END  
GO;  
  
EXECUTE uspOrderCost @orderID = 7;

### Podsumowania finansowe (Month/Kwartał/Year)

* procedury korzystające z odpowiednio miesiąca/kwartału/roku z daty i podsumowujące przychód z tego okresu

CREATE PROCEDURE *uspMonthlyFinanceSummary*(  
 @date date = null  
)  
AS  
BEGIN  
 IF @date IS NULL  
 BEGIN  
 SELECT 'data jest wymaganym parametrem'  
 END  
 ELSE  
 BEGIN  
 DECLARE @discount\_sum int, @non\_discount\_sum int;  
 SELECT @discount\_sum = SUM(Quantity \* OrderDetails.UnitPrice \* (1 - DiscountValue))  
 FROM OrderDetails  
 INNER JOIN Orders on Orders.OrderID = OrderDetails.OrderID  
 INNER JOIN Discounts on Orders.DiscountID = Discounts.DiscountID  
 WHERE Orders.DiscountID IS NOT NULL  
 AND YEAR(WhenToBeDone) = YEAR(@date)  
 AND MONTH(WhenToBeDone) = MONTH(@date)  
  
 SELECT @non\_discount\_sum = SUM(Quantity \* OrderDetails.UnitPrice)  
 FROM OrderDetails  
 INNER JOIN Orders on Orders.OrderID = OrderDetails.OrderID  
 WHERE Orders.DiscountID IS NULL  
 AND YEAR(WhenToBeDone) = YEAR(@date)  
 AND MONTH(WhenToBeDone) = MONTH(@date)  
  
 IF @discount\_sum IS NULL  
 BEGIN  
 SELECT @non\_discount\_sum as TotalMonthlyIncome  
 END  
 ELSE  
 BEGIN  
 IF @non\_discount\_sum IS NULL  
 BEGIN  
 SELECT @discount\_sum as TotalMonthlyIncome  
 END  
 ELSE  
 BEGIN  
 SELECT @non\_discount\_sum + @discount\_sum as TotalMonthlyIncome  
 END  
 END  
 END  
END  
GO;  
EXECUTE uspMonthlyFinanceSummary @date = '2023-01-12'

CREATE PROCEDURE *uspQuarterlyFinanceSummary*(  
 @date date = null  
)  
AS  
BEGIN  
 IF @date IS NULL  
 BEGIN  
 SELECT 'data jest wymaganym parametrem'  
 END  
 ELSE  
 BEGIN  
 DECLARE @discount\_sum int, @non\_discount\_sum int;  
 SELECT @discount\_sum = SUM(Quantity \* OrderDetails.UnitPrice \* (1 - DiscountValue))  
 FROM OrderDetails  
 INNER JOIN Orders on Orders.OrderID = OrderDetails.OrderID  
 INNER JOIN Discounts on Orders.DiscountID = Discounts.DiscountID  
 WHERE Orders.DiscountID IS NOT NULL  
 AND YEAR(WhenToBeDone) = YEAR(@date)  
 AND DATEPART(Quarter, WhenToBeDone) = DATEPART(Quarter, WhenToBeDone)  
  
 SELECT @non\_discount\_sum = SUM(Quantity \* OrderDetails.UnitPrice)  
 FROM OrderDetails  
 INNER JOIN Orders on Orders.OrderID = OrderDetails.OrderID  
 WHERE Orders.DiscountID IS NULL  
 AND YEAR(WhenToBeDone) = YEAR(@date)  
 AND DATEPART(Quarter, WhenToBeDone) = DATEPART(Quarter, WhenToBeDone)  
  
 IF @discount\_sum IS NULL  
 BEGIN  
 SELECT @non\_discount\_sum as QuarterlyFinanceSummary  
 END  
 ELSE  
 BEGIN  
 IF @non\_discount\_sum IS NULL  
 BEGIN  
 SELECT @discount\_sum as QuarterlyFinanceSummary  
 END  
 ELSE  
 BEGIN  
 SELECT @non\_discount\_sum + @discount\_sum as QuarterlyFinanceSummary  
 END  
 END  
 END  
END  
GO;  
EXECUTE uspQuarterlyFinanceSummary @date = '2023-01-12'

CREATE PROCEDURE *uspYearlyFinanceSummary*(  
 @date date = null  
)  
AS  
BEGIN  
 IF @date IS NULL  
 BEGIN  
 SELECT 'data jest wymaganym parametrem'  
 END  
 ELSE  
 BEGIN  
 DECLARE @discount\_sum int, @non\_discount\_sum int;  
 SELECT @discount\_sum = SUM(Quantity \* OrderDetails.UnitPrice \* (1 - DiscountValue))  
 FROM OrderDetails  
 INNER JOIN Orders on Orders.OrderID = OrderDetails.OrderID  
 INNER JOIN Discounts on Orders.DiscountID = Discounts.DiscountID  
 WHERE Orders.DiscountID IS NOT NULL  
 AND YEAR(WhenToBeDone) = YEAR(@date)  
  
 SELECT @non\_discount\_sum = SUM(Quantity \* OrderDetails.UnitPrice)  
 FROM OrderDetails  
 INNER JOIN Orders on Orders.OrderID = OrderDetails.OrderID  
 WHERE Orders.DiscountID IS NULL  
 AND YEAR(WhenToBeDone) = YEAR(@date)  
  
 IF @discount\_sum IS NULL  
 BEGIN  
 SELECT @non\_discount\_sum as YearlyFinanceSummary  
 END  
 ELSE  
 BEGIN  
 IF @non\_discount\_sum IS NULL  
 BEGIN  
 SELECT @discount\_sum as YearlyFinanceSummary  
 END  
 ELSE  
 BEGIN  
 SELECT @non\_discount\_sum + @discount\_sum as YearlyFinanceSummary  
 END  
 END  
 END  
END  
GO;  
EXECUTE uspYearlyFinanceSummary @date = '2023-01-12'

### Podsumowanie finansowe (zakres dat) - raport wew.

* procedura zlicza przychód uzyskany z tytułu sprzedaży produktów z danej kategorii w zadanym przedziale czasu

CREATE PROCEDURE *uspCategorySellSummary*(  
 @since date = '1900-01-01',  
 @to date = null  
)  
AS  
BEGIN  
 SELECT Categories.CategoryName, SUM(OrderDetails.Quantity \* Warehouse.UnitPrice) AS IncomeSum  
 FROM Categories  
 INNER JOIN Warehouse on Warehouse.CategoryID = Categories.CategoryID  
 INNER JOIN OrderDetails on Warehouse.MealID = OrderDetails.MealID  
 INNER JOIN Orders on OrderDetails.OrderID = Orders.OrderID  
 WHERE WhenToBeDone >= @since  
 AND WhenToBeDone <= @to  
 GROUP BY Categories.CategoryName  
END  
GO;  
EXECUTE uspCategorySellSummary @since = '2020-01-01', @to = '2022-12-01';  
EXECUTE uspCategorySellSummary @to = '2025-12-01';

### Podsumowanie finansowe (zakres dat) - raport wew.

* procedura podsumowuje uzyskany przychód z tytułu sprzedaży dla każdego produktu z osobna w zadanym przedziale czasu

CREATE PROCEDURE *uspProductSellSummary*(  
 @since date = '1900-01-01',  
 @to date  
)  
AS  
BEGIN  
 SELECT Warehouse.Name AS ProcutName, SUM(Quantity \* Warehouse.UnitPrice) AS IncomeSum  
 FROM Warehouse  
 INNER JOIN OrderDetails on Warehouse.MealID = OrderDetails.MealID  
 INNER JOIN Orders on OrderDetails.OrderID = Orders.OrderID  
 WHERE WhenToBeDone >= @since  
 AND WhenToBeDone <= @to  
 GROUP BY Warehouse.Name  
END  
GO;  
  
EXECUTE uspProductSellSummary @since = '2020-01-01', @to = '2022-12-01';  
EXECUTE uspProductSellSummary @to = '2025-12-01';

### Podsumowanie finansowe z zamówienia (OrderID) - faktura

* Procedura zwraca obecne w bazie dane niezbędne do wystawienia faktury

CREATE PROCEDURE *uspGetInvoice*(  
 @orderID int  
)  
AS  
BEGIN  
 DECLARE @depID int  
 SELECT @depID = DepID FROM Orders WHERE OrderID = @orderID  
 IF @depID IS NOT NULL  
 BEGIN  
 SELECT GETDATE() DataDzisiejsza,  
 CompanyName Nazwa,  
 NIP NIP,  
 Address Adres,  
 CityName Miasto,  
 PostalCode KodPocztowy,  
 WhenToBeDone DataWykonaniaUsługi,  
 Name NazwaPotrawy,  
 Quantity Ilość,  
 W.UnitPrice CenaZaSztukę,  
 DiscountValue Rabat  
 FROM Orders  
 INNER JOIN Departments D on Orders.DepID = D.DepID  
 INNER JOIN Cities C on C.CityID = D.CityID  
 INNER JOIN OrderDetails OD on Orders.OrderID = OD.OrderID  
 INNER JOIN Warehouse W on OD.MealID = W.MealID  
 INNER JOIN Companies C2 on C2.CompanyID = D.CompanyID  
 LEFT OUTER JOIN Discounts D2 on Orders.DiscountID = D2.DiscountID  
 WHERE Orders.OrderID = @orderID  
  
 END  
 ELSE  
 BEGIN  
 SELECT GETDATE() DataDzisiejsza,  
 FirstName Imie,  
 LastName Nazwisko,  
 WhenToBeDone DataWykonaniaUsługi,  
 Name NazwaPotrawy,  
 Quantity Ilość,  
 W.UnitPrice CenaZaSztukę,  
 DiscountValue Rabat  
 FROM Orders  
 INNER JOIN Customers Cu on Cu.CustomerID = Orders.CustomerID  
 INNER JOIN OrderDetails OD on Orders.OrderID = OD.OrderID  
 INNER JOIN Warehouse W on OD.MealID = W.MealID  
 LEFT OUTER JOIN Discounts D2 on Orders.DiscountID = D2.DiscountID  
 WHERE OD.OrderID = @orderID  
 END  
  
END  
GO;  
  
--przykladowe opdalenie  
EXECUTE uspGetInvoice @orderID = 8  
  
DECLARE @depID int  
SELECT @depID = CAST(DepID AS INT)  
FROM Orders  
WHERE @depID = DepID

### Podsumowanie finansowe z miesiąca(Firma ID, ClientID, miesiąc, rok) - faktura

Procedura zwraca dane do faktury za dany miesiąc dla danego klienta

1. CREATE PROCEDURE *uspGetMonthInvoice*(  
    @CustomerID INT = NULL,  
    @DepID INT = NULL,  
    @data date = NULL  
   ) AS  
   BEGIN  
    IF @data IS NULL OR (@CustomerID IS NULL AND @DepID IS NULL) OR (@CustomerID IS NOT NULL AND @DepID IS NOT NULL)  
    BEGIN  
    SELECT 'Wrong arguments'  
    END  
    ELSE  
    BEGIN  
    IF @DepID IS NOT NULL  
    BEGIN  
    SELECT GETDATE() DataDzisiejsza,  
    CompanyName Nazwa,  
    NIP NIP,  
    Address Adres,  
    CityName Miasto,  
    PostalCode KodPocztowy,  
    WhenToBeDone DataWykonaniaUsługi,  
    Name NazwaPotrawy,  
    Quantity Ilość,  
    W.UnitPrice CenaZaSztukę,  
    DiscountValue Rabat  
    FROM Orders  
    INNER JOIN Departments D on Orders.DepID = D.DepID  
    INNER JOIN Cities C on C.CityID = D.CityID  
    INNER JOIN OrderDetails OD on Orders.OrderID = OD.OrderID  
    INNER JOIN Warehouse W on OD.MealID = W.MealID  
    INNER JOIN Companies C2 on C2.CompanyID = D.CompanyID  
    LEFT OUTER JOIN Discounts D2 on Orders.DiscountID = D2.DiscountID  
    WHERE @DepID = Orders.DepID  
    AND YEAR(@data) = YEAR(Orders.WhenToBeDone)  
    AND MONTH(@data) = MONTH(Orders.WhenToBeDone)  
    END  
    ELSE  
    BEGIN  
    SELECT GETDATE() DataDzisjesza,  
    FirstName Imie,  
    LastName Nazwisko,  
    WhenToBeDone DataWykonaniaUsługi,  
    Name NazwaPotrawy,  
    Quantity Ilość,  
    W.UnitPrice CenaZeSztukę,  
    DiscountValue Rabat  
    FROM Orders  
    INNER JOIN Customers Cu on Cu.CustomerID = Orders.CustomerID  
    INNER JOIN OrderDetails OD on Orders.OrderID = OD.OrderID  
    INNER JOIN Warehouse W on OD.MealID = W.MealID  
    LEFT OUTER JOIN Discounts D2 on Orders.DiscountID = D2.DiscountID  
    WHERE @CustomerID = Orders.CustomerID  
    AND YEAR(@data) = YEAR(Orders.WhenToBeDone)  
    AND MONTH(@data) = MONTH(Orders.WhenToBeDone)  
     
     
    END  
    END  
   END  
   GO;  
   EXECUTE uspGetMonthInvoice @CustomerID = 1, @data = '2019-01-12 15:25:23.000'  
   EXECUTE uspGetMonthInvoice @CustomerID = 2, @data = '2022-01-12 15:25:25.000'  
   EXECUTE uspGetMonthInvoice @DepID = 2, @data = '2022-01-12 15:25:27.000'

### Widok Menu (kategoria, zakres cen)

* Procedura tworzy widok, wybierając z Menu tylko dania z kategorii podanej jako argument oraz z podanego jako argumenty przedziału cenowego, jeśli kategoria jest null’em bierzemy pod uwagę wszystkie kategorie, natomiast w przypadku nulla, w którejkolwiek z granic cenowych bierzemy pod uwagę cały zakres.
* Widok ten może wyświetlić Waiter (aby lepiej doradzić klientowi) oraz Manager i WarehouseManager.
* Procedura przyjmuje 3 argumenty: kategorię do wyświetlenia, dolną granicę cenową oraz górną granicę cenową.
* Procedura generuje widok, więc nie zwraca żadnych parametrów wyjściowych.

CREATE PROCEDURE *uspMenuSpec*(  
 @chosenCategory AS VARCHAR(60) = NULL,  
 @minPrice AS INT = NULL,  
 @maxPrice AS INT = NULL  
)  
AS  
BEGIN  
 IF @chosenCategory IS NULL  
 BEGIN  
 IF @minPrice IS NULL OR @maxPrice IS NULL  
 BEGIN  
 SELECT W.Name, W.UnitPrice, C.CategoryName  
 FROM Menu  
 INNER JOIN Warehouse W on W.MealID = Menu.MealID  
 INNER JOIN Categories C on C.CategoryID = W.CategoryID  
 END  
 ELSE  
 BEGIN  
 SELECT W.Name, W.UnitPrice, C.CategoryName  
 FROM Menu  
 INNER JOIN Warehouse W on W.MealID = Menu.MealID  
 INNER JOIN Categories C on C.CategoryID = W.CategoryID  
 WHERE W.UnitPrice BETWEEN @minPrice AND @maxPrice  
 END  
 END  
 ELSE  
 BEGIN  
 IF @minPrice IS NULL OR @maxPrice IS NULL  
 BEGIN  
 SELECT W.Name, W.UnitPrice, C.CategoryName  
 FROM Menu  
 INNER JOIN Warehouse W on W.MealID = Menu.MealID  
 INNER JOIN Categories C on C.CategoryID = W.CategoryID  
 WHERE C.CategoryName = @chosenCategory  
 END  
 ELSE  
 BEGIN  
 SELECT W.Name, W.UnitPrice, C.CategoryName  
 FROM Menu  
 INNER JOIN Warehouse W on W.MealID = Menu.MealID  
 INNER JOIN Categories C on C.CategoryID = W.CategoryID  
 WHERE (W.UnitPrice BETWEEN @minPrice AND @maxPrice)  
 AND C.CategoryName = @chosenCategory  
 END  
 END  
END  
  
EXECUTE uspMenuSpec  
-- @chosenCategory,  
 @minPrice = 0,  
 @maxPrice = 10000;

### Ilu klientów indywidualnych było w restauracji co najmniej x razy (x)

* Procedura zlicza ilu klientów indywidualnych odwiedziło restauracje (w sensie złożenia zamówienia) więcej niż podana jako argument liczba razy.
* Procedurę mogą wywołać Manager i Accountant.
* Procedura przyjmuje jeden argument, który jest minimalną liczbą wizyt klienta w restauracji, potrzebną do uwzględnienia go w wyniku. W razie wywołania procedury z wartością null, zwrócona zostanie ilość wszystkich klientów.
* Procedura zwraca parametr wyjściowy w postaci liczby klientów spełniających opisane wymaganie.

CREATE PROCEDURE *uspHowManyVisitors*(  
 @howManyTimes AS INT = NULL,  
 @howManyVisitors INT OUTPUT  
) AS  
 BEGIN  
 IF @howManyTimes IS NULL  
 BEGIN  
 SELECT CustomerID FROM Customers;  
 END  
 ELSE  
 BEGIN  
 SELECT CustomerID  
 FROM Orders  
 GROUP BY CustomerID  
 HAVING COUNT(CustomerID) >= @howManyTimes  
 AND CustomerID IS NOT NULL  
 END  
 SELECT @howManyVisitors = @@ROWCOUNT;  
 END  
GO;  
  
DECLARE @howManyTimes INT, @result INT;  
SET @howManyTimes = 2;  
EXECUTE uspHowManyVisitors  
 @howManyTimes,  
 @howManyVisitors = @result OUTPUT;  
SELECT @result as result

### Ile razy dany klient indywidualny był w restauracji (klientID)

* Procedura zlicza ile razy dany klient indywidualny odwiedził restauracje.
* Procedurę mogą wywołać Manager i Accountant.
* Procedura przyjmuje jeden argument: identyfikator w bazie klienta, o którym chcemy uzyskać ww. informacje, w przypadku wywołania procedury z wartością null, zwrócone zostanie -1 (może być i inf o błędzie, ale jeszcze nie wiem czy bd umiał to obsłużyć)-> z tego co wyczytałem to TRY/CATCH nie łapie takich rzeczy, ale jakbyście coś wiedzieli to dawajcie znać.
* Procedura zwraca parametr wyjściowy w postaci liczby wizyt danego klienta w restauracji.

CREATE PROCEDURE *uspHowManyVisits*(  
 @clientID AS INT,  
 @howManyVisits AS INT OUTPUT  
) AS  
BEGIN  
 IF @clientID IS NULL  
 BEGIN  
 SELECT @howManyVisits = -1;  
 END  
 ELSE  
 BEGIN  
 SET @howManyVisits = 0;  
 SELECT @howManyVisits = @howManyVisits + 1 --idk czy zadziała, ale mam w głowie alternatywę  
 FROM Orders O  
 WHERE O.CustomerID = @clientID  
 END  
END  
GO;  
  
DECLARE @result2 INT;  
EXECUTE uspHowManyVisits  
 @clientID = 7,  
 @howManyVisits = @result2 OUTPUT;  
SELECT @result2 AS 'how many visits by client 1';

### Średnia ilość obsłużonych zamówień/dzień (Kelner)

* Procedura oblicza średnią ilość zamówień obsłużonych przez kelnera na dzień w ciągu całej jego pracy w restauracji.
* Procedurę może wywołać Manager.
* Procedura przyjmuje jeden argument: identyfikator wybranego kelnera. Jeśli podany zostanie argument w postaci null’a, zwrócone zostanie -1 (może być i inf o błędzie, ale jeszcze nie wiem czy bd umiał to obsłużyć) -> jak w 9..
* Procedura zwraca jeden parametr wyjściowy: liczbę reprezentującą ww. średnią.

CREATE PROCEDURE *uspAverageOrdersServiced*(  
 @waiterID AS INT,  
 @averageAmount AS DECIMAL(10, 5) OUTPUT  
) AS  
BEGIN  
 DECLARE @howManyDays INT;  
 SET @howManyDays = 0;  
 IF @waiterID IS NULL  
 BEGIN  
 SELECT @averageAmount = -1;  
 END  
 ELSE  
 BEGIN  
 DECLARE @jobFromID VARCHAR(100);  
  
 SELECT @jobFromID = J.JobName  
 FROM Employees E  
 INNER JOIN Jobs J ON J.JobID = E.JobID  
 WHERE EmployeeID = @waiterID  
  
 IF @jobFromID != 'Waiter'  
 BEGIN  
 SELECT @averageAmount = -10;  
 END  
 ELSE  
 BEGIN  
 DECLARE @howManyOrders INT;  
 SET @howManyOrders = 0;  
  
 SELECT OrderID  
 FROM Orders  
 WHERE EmployeeID = @waiterID  
  
  
 BEGIN  
 SELECT @howManyOrders = @@ROWCOUNT  
 BEGIN  
 SELECT @howManyDays = DATEDIFF(weekday, HireDate, GETDATE())  
 FROM Employees  
 WHERE EmployeeID = @waiterID  
 END  
 SELECT @averageAmount =  
 CAST(@howManyOrders AS DECIMAL(10, 5)) / CAST(@howManyDays AS DECIMAL(10, 5))  
 END  
 END  
 END  
END  
GO;  
  
DECLARE @result3 DECIMAL(10, 5);  
EXECUTE uspAverageOrdersServiced  
 @waiterID = 3,  
 @averageAmount = @result3 OUTPUT;  
SELECT @result3 AS 'Average orders per day for waiter with employeeID = @waiterID';

### Jak często jest wybierany dany typ stolika (typ stolika)

* Umożliwia otrzymanie informacji ile razy dany typ stolika był wybierany
* Może go wyświetlić Waiter, Accountant oraz Chef.
* Przyjmuje typ stolika
* Zwraca ile razy był wybrany

CREATE PROCEDURE *uspUsageOfTableTypes*(  
 @TClassName AS nvarchar(100),  
 @howManyTimesChosen INT OUTPUT  
) AS  
BEGIN  
 SELECT @howManyTimesChosen = COUNT(TU.TableID)  
 FROM TablesUsed TU  
 INNER JOIN Tables T on T.TableID = TU.TableID  
 INNER JOIN TableClasses TC on TC.TClassID = T.TClassID  
 WHERE TC.TClassName = @TClassName  
 GROUP BY TC.TClassID, TC.TClassName  
END  
GO;  
  
DECLARE @result as INT;  
EXECUTE uspUsageOfTableTypes  
 @TClassName = 'Super',  
 @howManyTimesChosen = @result OUTPUT;  
SELECT @result as howManyTimesChosen

### Który stolik jest wybierany najczęściej (typ stolika)

* Umożliwia otrzymanie informacji który stolik danego typu był wybierany najczęściej
* Może go wyświetlić Waiter, Accountant oraz Chef.
* Przyjmuje typ stolika
* Zwraca numer stolika

CREATE PROCEDURE *uspMostUsedTable*(  
 @TClassName AS nvarchar(100),  
 @whichTable INT OUTPUT,  
 @howMany INT OUTPUT  
)  
AS  
BEGIN  
 SELECT TOP 1 @whichTable = TU.TableID, @howMany = COUNT(TU.TableID)  
 FROM TablesUsed TU  
 INNER JOIN Tables T on T.TableID = TU.TableID  
 INNER JOIN TableClasses TC on TC.TClassID = T.TClassID  
 WHERE TC.TClassName = @TClassName  
 GROUP BY TU.TableID, TC.TClassName  
 ORDER BY count(TU.TableID) desc  
END  
GO;  
  
DECLARE @result as INT;  
DECLARE @result1 as INT;  
EXECUTE uspMostUsedTable  
 @TClassName = 'Super',  
 @whichTable = @result OUTPUT,  
 @howMany = @result1 OUTPUT;  
SELECT @result as TableID, @result1 as howManyTimesChosen

### Jakie zniżki ma dany klient (numer klienta)

* Umożliwia sprawdzenie jakie zniżki obecnie dostępne posiada klient
* Może go wyświetlić Waiter oraz Accountant.
* Przyjmuje ID klienta
* Zwraca dostępne zniżki

CREATE PROCEDURE *uspAllClientDiscounts*(  
 @clientID AS INT = NULL  
) AS  
BEGIN  
 DECLARE @clientDoesntExist VARCHAR(10)  
 SELECT @clientDoesntExist = IIF(@clientID NOT IN (SELECT CustomerID FROM Discounts), 'true', 'false')  
  
 IF @clientID IS NULL OR @clientDoesntExist = 'true'  
 BEGIN  
 PRINT ('Invalid argument was passed or client doesnt have a discount!')  
 END  
 ELSE  
 BEGIN  
 SELECT DiscountID, DiscountType, DiscountValue  
 FROM Discounts  
 WHERE CustomerID = @clientID  
 AND IfUsed = 0  
 AND Until >= GETDATE()  
 END  
END  
GO;  
--przykładowe odpalenie  
EXECUTE uspAllClientDiscounts  
 @clientID = 5;

### Aktualność menu

* Wyświetla które pozycje z menu są przestarzałe
* Może go wyświetlić Waiter oraz Chef.
* Wyświetla listę dań

CREATE PROCEDURE *uspMaxTwoWeeksMenu*AS  
 BEGIN  
 DECLARE @twoWeeksOrOlder INT, @allDishes INT;  
 SET @twoWeeksOrOlder = 0;  
 SET @allDishes = 0;  
  
 SELECT MealID FROM Menu  
  
 SELECT @allDishes = @@ROWCOUNT  
  
 SELECT MealID FROM Menu WHERE DATEDIFF(day, Menu.SinceDate, GETDATE()) >= 13 --13 bo robimy to dzień przed  
  
 SELECT @twoWeeksOrOlder = @@ROWCOUNT  
  
 IF 2\* @twoWeeksOrOlder >= @allDishes  
 BEGIN  
 PRINT('I don t know how to delete dishes from the Menu but you definitely should!') -- Outdated:')--, @twoWeeksOrOlder, 'of', @allDishes)  
 SELECT MealID FROM Menu WHERE DATEDIFF(day, Menu.SinceDate, GETDATE()) >= 13 --13 bo robimy to dzień przed  
 END  
 ELSE  
 BEGIN  
 PRINT('Good job , everything on point, how unusual!') -- Outdated:')--, @twoWeeksOrOlder, 'of', @allDishes)  
 SELECT MealID FROM Menu WHERE DATEDIFF(day, Menu.SinceDate, GETDATE()) >= 13 --13 bo robimy to dzień przed  
 END  
 END  
 GO;  
  
EXECUTE uspMaxTwoWeeksMenu

### Rezerwacje w danym tygodniu

* Umożliwia wyświetlenie wszystkich rezerwacji z danego tygodnia
* Może go wyświetlić Waiter oraz Accountant.
* Przyjmuje numer tygodnia oraz rok
* Zwraca liczbę rezerwacji

CREATE PROCEDURE *uspWeeklyTableReservations*(  
 @week\_nb AS DATETIME = NULL,  
 @year AS INT = NULL  
)  
AS  
BEGIN  
 IF @week\_nb IS NULL  
 SET @week\_nb = (SELECT DATEPART(iso\_week, GETDATE()))  
 IF @year IS NULL  
 SET @year = (SELECT YEAR(GETDATE()))  
 SELECT O.OrderID, TableID, WhenOrdered 'Reservation date', WhenToBeDone 'Reserved date'  
 FROM Orders O  
 INNER JOIN TablesUsed TU ON O.OrderID = TU.OrderID  
 WHERE ForHowLong > '00:00:00'  
 AND SeatsNeeded > 0  
 AND @week\_nb = DATEPART(iso\_week, WhenToBeDone)  
 AND @year = YEAR(WhenToBeDone)  
END  
GO;  
EXECUTE uspWeeklyTableReservations @week\_nb = 3

### Rezerwacje w danym miesiącu

* Umożliwia wyświetlenie wszystkich rezerwacji z danego miesiąca
* Może go wyświetlić Waiter oraz Accountant.
* Przyjmuje numer miesiąca oraz rok
* Zwraca liczbę rezerwacji

CREATE PROCEDURE *uspMonthlyTableReservations*(  
 @month\_nb AS DATETIME = NULL,  
 @year AS INT = NULL  
)  
AS  
BEGIN  
 IF @month\_nb IS NULL  
 SET @month\_nb = (SELECT DATEPART(iso\_week, GETDATE()))  
 IF @year IS NULL  
 SET @year = (SELECT YEAR(GETDATE()))  
 SELECT O.OrderID, TableID, WhenOrdered 'Reservation date', WhenToBeDone 'Reserved date'  
 FROM Orders O  
 INNER JOIN TablesUsed TU ON O.OrderID = TU.OrderID  
 WHERE ForHowLong > '00:00:00'  
 AND SeatsNeeded > 0  
 AND @month\_nb = DATEPART(month, WhenToBeDone)  
 AND @year = YEAR(WhenToBeDone)  
END  
GO;  
EXECUTE uspMonthlyTableReservations @year = 2022

### Rabaty w danym tygodniu

* Umożliwia wyświetlenie ilości użytych rabatów w danym tygodniu
* Może go wyświetlić Waiter oraz Accountant.
* Przyjmuje numer tygodnia oraz rok
* Zwraca liczbę rabatów

CREATE PROCEDURE *uspWeeklyDiscounts*(  
 @week\_nb AS DATETIME = NULL,  
 @year AS INT = NULL  
)  
AS  
BEGIN  
 IF @week\_nb IS NULL  
 SET @week\_nb = (SELECT DATEPART(iso\_week, GETDATE()))  
 IF @year IS NULL  
 SET @year = (SELECT YEAR(GETDATE()))  
 SELECT WhenToBeDone,  
 O.DiscountID,  
 DiscountValue 'DiscountValue[%]',  
 (  
 SELECT SUM(Quantity \* OD.UnitPrice) - SUM(Quantity \* OD.UnitPrice \* (1 - Dis2.DiscountValue))  
 FROM OrderDetails OD  
 INNER JOIN Orders O2 ON O2.OrderID = OD.OrderID  
 INNER JOIN Discounts Dis2 ON Dis2.DiscountID = O2.DiscountID  
 WHERE OD.OrderID = O.OrderID  
 ) 'DiscountedValue[money]'  
 FROM Orders O  
 INNER JOIN Discounts Dis on Dis.DiscountID = O.DiscountID  
 WHERE O.DiscountID IS NOT NULL  
 AND DATEPART(iso\_week, WhenToBeDone) = @week\_nb  
 AND YEAR(WhenToBeDone) = @year  
END  
GO;  
EXECUTE uspWeeklyDiscounts

### Rabaty w danym miesiącu

* Umożliwia wyświetlenie ilości użytych rabatów w danym miesiącu
* Może go wyświetlić Waiter oraz Accountant.
* Przyjmuje numer miesiąca oraz rok
* Zwraca liczbę rabatów

CREATE PROCEDURE *uspMonthlyDiscounts*(  
 @month\_nb AS DATETIME = NULL,  
 @year AS INT = NULL  
)  
AS  
BEGIN  
 IF @month\_nb IS NULL  
 SET @month\_nb = (SELECT DATEPART(month , GETDATE()))  
 IF @year IS NULL  
 SET @year = (SELECT YEAR(GETDATE()))  
 SELECT WhenToBeDone,  
 O.DiscountID,  
 DiscountValue 'DiscountValue[%]',  
 (  
 SELECT SUM(Quantity \* OD.UnitPrice) - SUM(Quantity \* OD.UnitPrice \* (1 - Dis2.DiscountValue))  
 FROM OrderDetails OD  
 INNER JOIN Orders O2 ON O2.OrderID = OD.OrderID  
 INNER JOIN Discounts Dis2 ON Dis2.DiscountID = O2.DiscountID  
 WHERE OD.OrderID = O.OrderID  
 ) 'DiscountedValue[money]'  
 FROM Orders O  
 INNER JOIN Discounts Dis on Dis.DiscountID = O.DiscountID  
 WHERE O.DiscountID IS NOT NULL  
 AND DATEPART(month, WhenToBeDone) = @month\_nb  
 AND YEAR(WhenToBeDone) = @year  
END  
GO;  
EXECUTE uspMonthlyDiscounts