**Zapytania SELECT**

1. **Najbardziej Dochodowy Klient**: Napisz zapytanie, które znajdzie wszystkich klientów, którzy zamówili te same produkty, co najbardziej dochodowy klient. Ustal najbardziej dochodowego klienta na podstawie całkowitej wartości zamówień.

with temp as (

select distinct od.ProductID, o.CustomerID

from [Order Details] as od

inner join Orders as o

on od.OrderID = o.OrderID

where o.CustomerID = (select CustomerID from (

select top 1 o.CustomerID,

round(cast(sum(od.UnitPrice \* od.Quantity \* (1 - od.Discount)) as money), 2) as TotalSum

from [Order Details] as od

inner join Orders as o

on od.OrderID = o.OrderID

group by o.CustomerID

order by TotalSum desc

) as temp

)

)

select distinct o.CustomerID

from [Order Details] as od

inner join Orders as o

on od.OrderID = o.OrderID

where od.ProductID in (select ProductID from temp)

and o.CustomerID != (select top 1 CustomerID from temp)

2. **Niezamówione Produkty**: Znajdź produkty, które nie były nigdy zamówione, ale są w tych samych kategoriach, co produkty, które były zamówione.

with temp as (

select distinct od.ProductID, p.CategoryID

from [Order Details] as od

inner join Products as p

on od.ProductID = p.ProductID

)

select ProductID

from Products

where ProductID not in (select ProductID from temp)

and CategoryID in (select distinct CategoryID from temp)

3. **Pracownicy Powyżej Średniej**: Używając okienkowych funkcji SQL, znajdź pracowników, którzy osiągnęli więcej niż średnia sprzedaż w ich regionie.

with EmployeeRegion as (

select distinct e.EmployeeID, r.RegionID

from Employees as e

inner join EmployeeTerritories as et

on e.EmployeeID = et.EmployeeID

inner join Territories as t

on et.TerritoryID = t.TerritoryID

inner join Region as r

on t.RegionID = r.RegionID

),

RegionAvgSale as (

select distinct er.RegionID,

round(cast(avg(od.UnitPrice \* od.Quantity \* (1 - od.Discount)) over (partition by er.RegionID) as money), 2) as AvgSale

from [Order Details] as od

inner join Orders as o

on od.OrderID = o.OrderID

inner join EmployeeRegion as er

on o.EmployeeID = er.EmployeeID

),

EmployeeAvgSale as (

select distinct er.EmployeeID,

round(cast(avg(od.UnitPrice \* od.Quantity \* (1 - od.Discount)) over (partition by er.EmployeeID) as money), 2) as AvgSale

from [Order Details] as od

inner join Orders as o

on od.OrderID = o.OrderID

inner join EmployeeRegion as er

on o.EmployeeID = er.EmployeeID

)

select er.EmployeeID

from EmployeeRegion as er

inner join RegionAvgSale as ras

on er.RegionID = ras.RegionID

inner join EmployeeAvgSale as eas

on er.EmployeeID = eas.EmployeeID

where eas.AvgSale > ras.AvgSale

order by er.EmployeeID asc

4. **Procent Sprzedaży od Dostawcy**: Napisz zapytanie, które zestawi, jaki procent sprzedaży przypada na każdego dostawcę.

declare @totalSum money;

select @totalSum = round(cast(sum(UnitPrice \* Quantity \* (1 - Discount)) as money), 2) from [Order Details];

select EmployeeID, SumSales, (SumSales \* 100 / @totalSum) as SalesPercent

from (

select distinct o.EmployeeID,

round(cast(sum(od.UnitPrice \* od.Quantity \* (1 - od.Discount)) as money), 2) as SumSales

from [Order Details] as od

inner join Orders as o

on od.OrderID = o.OrderID

group by o.EmployeeID

) as temp;

5. **Najbardziej Zyskowne Kategorie**: Znajdź trzy najbardziej zyskowne kategorie produktów dla każdego roku.

**Procedury**

1. **Klienci Pracownika**: Procedura, która przyjmuje identyfikator pracownika i zwraca wszystkich klientów, którzy dokonali u niego zakupów, łącząc tabele `Orders` i `Customers`.

create proc p1 @employeeID int

as

begin

select distinct o.CustomerID, c.CompanyName

from Orders as o

inner join Customers as c

on o.CustomerID = c.CustomerID

where o.EmployeeID = @employeeID;

end;

go

exec p1 2;

2. **Dostawcy w Kraju**: Procedura, która dla danego kraju znajduje dostawców i sumę wartości ich dostaw, łącząc tabele `Suppliers` i `Products`.

create proc p2 @country nvarchar(15)

as

select s.SupplierID, s.CompanyName, sum(p.UnitPrice \* p.UnitsOnOrder) as TotalSum

from Products as p

inner join Suppliers as s

on p.SupplierID = s.SupplierID

where s.Country = @country

group by s.SupplierID, s.CompanyName;

go

exec p2 N'UK';

3. \*\*Miesięczny Raport\*\*: Procedura, która zwraca miesięczny raport sprzedaży. Powinna łączyć tabelę zamówień z tabelą pracowników i produktów.

4. \*\*Przypisanie Zamówień\*\*: Procedura, która przydziela nowe zamówienia do pracowników w zależności od ich obecnych obciążeń (na podstawie liczby obsługiwanych obecnie zamówień).

5. \*\*Raport Roczny\*\*: Procedura do generowania zestawienia rocznej sprzedaży według krajów i kategorii produktów, korzystając z tabel `Orders`, `Order Details`, `Customers` i `Products`.

**Funkcje Inline**

1. **Historia Zakupów Klienta**: Funkcja, która dla podanego klienta zwraca jego historię zakupów, korzystając z tabel `Orders` i `Order Details`.

create function if1(@id nchar(5))

returns table

as

return (

select o.OrderID, od.ProductID, od.UnitPrice, od.Quantity, od.Discount

from Orders as o

inner join [Order Details] as od

on o.OrderID = od.OrderID

where o.CustomerID = @id

);

go

select \* from if1('BOLID');

2. Popularna Kategoria w Kraju: Funkcja, która zwraca najpopularniejszą kategorię produktów w danym kraju, łącząc tabelę `Orders` z tabelą `Products`.

create function if2(@country nvarchar(15))

returns table

as

return (

select CategoryID

from (

select top 1 o.ShipCountry, p.CategoryID, count(1) as TotalQty

from Orders as o

inner join [Order Details] as od

on o.OrderID = od.OrderID

inner join Products as p

on od.ProductID = p.ProductID

where o.ShipCountry = @country

group by o.ShipCountry, p.CategoryID

order by o.ShipCountry asc, TotalQty desc

) as t

);

go

select \* from f2(N'Austria');

3. **Top 3 Produktów Dostawcy**: Funkcja, która zwraca trzy najczęściej zamawiane produkty od danego dostawcy.

create function if3(@supplierID int)

returns @temp table (ProductID int)

as

begin

insert into @temp

select top 3 p.ProductID

from [Order Details] as od

inner join Products as p

on od.ProductID = p.ProductID

where p.SupplierID = 8

group by p.ProductID

order by count(od.ProductID) desc;

return;

end;

go

select \* from if3(8);

4. \*\*Dostawcy z Wieloma Produktami\*\*: Funkcja, która zwraca wszystkich dostawców, którzy dostarczyli więcej niż 10 różnych produktów.

5. \*\*Obroty Klienta\*\*: Funkcja, która dla danego klienta oblicza jego całkowite obroty, korzystając z tabel `Orders` i `Order Details`.

**Funkcje Skalarne**

1. **Najbardziej Zyskowny Klient w Kwartale**: Funkcja, która zwraca identyfikator najbardziej zyskownego klienta w danym kwartale roku.

create function sf1(

@year smallint,

@quarter smallint

)

returns nchar(5)

as

begin

declare @client nchar(5);

select top 1 @client = CustomerID from (

select CustomerID, [Year], [Quarter],

sum(RowSum) as TotalSum

from (

select o.CustomerID, year(o.OrderDate) as [Year],

case

when month(o.OrderDate) between 1 and 3 then 1

when month(o.OrderDate) between 4 and 6 then 2

when month(o.OrderDate) between 7 and 9 then 3

when month(o.OrderDate) between 10 and 12 then 4

end as [Quarter],

round(cast(od.UnitPrice \* od.Quantity \* (1 - od.Discount) as money), 2) as RowSum

from Orders as o

inner join [Order Details] as od

on o.OrderID = od.OrderID

) as temp

group by CustomerID, [Year], [Quarter]

) as t

where [Year] = @year and [Quarter] = @quarter

order by [Year] asc, [Quarter] asc, TotalSum desc;

return @client;

end;

go

select dbo.sf1(1997, 3) as Client

2. **Średni Czas Dostawy**: Funkcja skalarna, która obliczy średnią liczbę dni od momentu złożenia zamówienia do jego wysyłki na podstawie wszystkich zamówień w bazie danych dla danego roku..

create function sf2(@year int)

returns int

as

begin

declare @result int;

select @result = avg(datediff(day, OrderDate, ShippedDate))

from Orders

where year(OrderDate) = @year;

return @result;

end;

go

select dbo.sf2(1997) as Result;

3. **Dochodowa Kategoria dla Pracownika**: Funkcja, która zwraca najbardziej dochodową kategorię produktów dla danego pracownika.

create function sf3(@employee int)

returns int

as

begin

declare @categoryID int;

select top 1 @categoryID = p.CategoryID

from [Order Details] as od

inner join Orders as o

on od.OrderID = o.OrderID

inner join Products as p

on od.ProductID = p.ProductID

where o.EmployeeID = 1

group by p.CategoryID

order by cast(round(sum((od.UnitPrice \* od.Quantity \* (1 - od.Discount))), 2) as money) desc;

return @categoryID;

end;

go

select dbo.sf3(1) as Result

4. **Koszty Wysyłki do Klienta**: Funkcja, która zwraca sumę kosztów wysyłki dla danego klienta w danym roku.

create function sf4(

@clientID nchar(5),

@year int

)

returns money

as

begin

declare @totalSum money;

select @totalSum = cast(round(sum(Freight), 2) as money)

from Orders

where CustomerID = @clientID

and year(OrderDate) = @year;

return @totalSum;

end;

go

select dbo.sf4('VINET', 1996) as Result

5. \*\*Sprzedaż w Kategorii\*\*: Funkcja, która oblicza, jaki procent całkowitej sprzedaży firmy przypada na daną kategorię produktów.

**Triggery**

1. **Cenowy Alarm**: Utwórz trigger, który będzie monitorować tabelę `Products` i generować wpis w nowej tabeli `PriceAlerts` (AlertID, ProductID, OldPrice, NewPrice, ChangedDate) jeśli cena produktu zostanie podniesiona o więcej niż 10%.

create table PriceAlerts (

AlertID int identity(1, 1) primary key,

ProductID int not null,

OldPrice money not null,

NewPrice money not null,

ChangedDate datetime not null

)

go

create index idx\_productid on PriceAlerts (ProductID);

go

create trigger t1

on Products

after update

as

begin

declare @oldPrice money, @newPrice money;

select @oldPrice = UnitPrice from deleted;

select @newPrice = UnitPrice from inserted;

if (@newPrice \* 100 / @oldPrice) > 110

begin

insert into PriceAlerts (ProductID, OldPrice, NewPrice, ChangedDate)

values ((select ProductID from inserted), @oldPrice, @newPrice, getdate());

end;

end;

go

2. \*\*Monitorowanie Zmian Zamówień\*\*:

Utwórz trigger dla tabeli `Orders`, który zapisze w nowo utworzonej tabeli `OrderAudit` wszystkie zmiany wartości zamówień. Powinien on przechowywać informacje o tym, co zostało zmienione, kiedy i przez kogo.

3. \*\*Walidacja Stanu Magazynowego\*\*:

Utwórz trigger dla tabeli `Order Details`, który nie pozwoli na dodanie zamówienia, jeżeli nie ma wystarczającej ilości produktu na stanie.

4. \*\*Automatyczna Premia dla Pracownika\*\*:

Za każdym razem, gdy w tabeli `Orders` zostanie dodane nowe zamówienie, trigger powinien sprawdzić, czy pracownik, który go zrealizował, osiągnął próg 100 zamówień. Jeżeli tak, powinien automatycznie generować wpis w tabeli `EmployeeBonuses`.

5. \*\*Zachowanie Spójności Danych\*\*:

W przypadku usunięcia klienta z tabeli `Customers`, trigger powinien automatycznie anulować wszystkie niezrealizowane zamówienia dla tego klienta w tabeli `Orders` i zalogować te anulowania w tabeli `CancelledOrders`.