Implementatieplan

Groep 3c

**RIM Outdoor Paradise**

**CREATE** **TABLE** Position **(**

pos\_numb **int** **IDENTITY(**1**,**1**) NOT** **NULL,**

pos\_abbr **Varchar(**20**),**

pos\_title **Varchar(**50**),**

min\_salary **float,**

max\_salary **float,**

**PRIMARY** **KEY** **(**pos\_numb**)**

**)**

**CREATE** **TABLE** Target **(**

target\_code **int** **IDENTITY(**1**,**1**) NOT** **NULL,**

target\_description\_en **Varchar(**255**)** **NOT** **NULL,**

**PRIMARY** **KEY** **(**target\_code**)**

**)**

**CREATE** **TABLE** Order\_method **(**

order\_method\_code **int** **IDENTITY(**1**,**1**) NOT** **NULL,**

order\_method\_en **Varchar(**255**)** **NOT** **NULL,**

**PRIMARY** **KEY** **(**order\_method\_code**)**

**)**

**CREATE** **TABLE** Country **(**

country\_code **int** **IDENTITY(**1**,**1**)** **NOT** **NULL,**

country **Varchar(**50**)** **NOT** **NULL,**

minimum\_wage **Float** **NOT** **NULL** **CHECK([**minimum\_wage**]** **>** 0**)**

**[language]** **Varchar(**50**)** **NOT** **NULL,**

**PRIMARY** **KEY** **(**country\_code**)**

**)**

**CREATE** **TABLE** Promotion **(**

pr\_number **Int IDENTITY(**1**,**1**)** **NOT** **NULL,**

date\_start **DateTime,**

date\_end **DateTime,**

**[**description**]** **Varchar(**255**),**

**PRIMARY** **KEY** **(**pr\_number**)**

**)**

**CREATE** **TABLE** Product\_line **(**

prod\_line\_code **int** **IDENTITY(**1**,**1**) NOT** **NULL,**

prod\_line\_en **Varchar(**80**)** **NOT** **NULL,**

**PRIMARY** **KEY** **(**prod\_line\_code**)**

**)**

**CREATE** **TABLE** Return\_reason **(**

return\_reason\_code **int** **IDENTITY(**1**,**1**) NOT** **NULL,**

return\_description\_en **Varchar(**50**)** **NOT** **NULL,**

**PRIMARY** **KEY** **(**return\_reason\_code**)**

**)**

**CREATE** **TABLE** Retailer\_type **(**

retailer\_type\_code **int** **IDENTITY(**1**,**1**) NOT** **NULL,**

type\_name\_en **Varchar(**255**)** **NOT** **NULL,**

**PRIMARY** **KEY** **(**retailer\_type\_code**)**

**)**

**CREATE** **TABLE** Order\_finance **(**

finance\_code **int** **IDENTITY(**1**,**1**)** **NOT** **NULL,**

finance\_type **Varchar(**10**)** **NOT** **NULL** **CHECK([**finance\_type**]** **IN** **(**'expense'**,** 'revenue'**)),**

finance\_description **Varchar(**255**),**

**PRIMARY** **KEY** **(**finance\_code**)**

**)**

**CREATE** **TABLE** Vacation **(**

vac\_id **int** **IDENTITY(**1**,**1**)** **NOT** **NULL,**

transport **Varchar(**255**),**

duration **int** **NOT** **NULL** **CHECK([**duration**]** **>** 0**),**

price **float** **NOT** **NULL** **CHECK([**price**]** **>** 0**),**

min\_people **int** **NOT** **NULL,**

max\_people **int** **NOT** **NULL,**

child\_proof **bit** **NOT** **NULL,**

product\_type\_code **INT,**

**PRIMARY** **KEY** **(**vac\_id**)**

**)**

**CREATE** **TABLE** Bonus **(**

bonus\_date **DATE** **NOT** **NULL,**

bonus\_amo **Float** **NOT** **NULL** **CHECK([**bonus\_amo**]** **>=** 0**),**

emp\_id **int** **NOT** **NULL,**

**CONSTRAINT** **[**PK\_BONUS**]** **PRIMARY** **KEY** CLUSTERED **([**emp\_id**]** **ASC,** **[**bonus\_date**]** **ASC),**

**FOREIGN** **KEY** **(**emp\_id**)** **REFERENCES** Employee**(**emp\_id**)**

**)**

**CREATE** **TABLE** Training **(**

training\_id **int** **IDENTITY(**1**,**1**) NOT** **NULL,**

course **Varchar(**255**),**

pos\_numb **int,**

**PRIMARY** **KEY** **(**training\_id**),**

**FOREIGN** **KEY** **(**pos\_numb**)** **REFERENCES** Training**(**pos\_numb**)**

**)**

**CREATE** **TABLE** Emp\_training**(**

training\_id **int** **NOT** **NULL,**

emp\_id **int** **NOT** **NULL,**

**[year]** **int** **NOT** **NULL,**

**CONSTRAINT** **[**EMPLOYEE\_VOLGT**]** **PRIMARY** **KEY** CLUSTERED **([**training\_id**]** **ASC,** **[**emp\_id**]** **ASC,** **[date]** **ASC),**

**FOREIGN** **KEY** **(**training\_id**)** **REFERENCES** Training**(**training\_id**),**

**FOREIGN** **KEY** **(**emp\_id**)** **REFERENCES** Employee**(**emp\_id**)**

**)**

**CREATE** **TABLE** Pref\_appraisal **(**

target\_code **int** **NOT** **NULL,**

**[**description**]** **Varchar(**255**)** **NOT** **NULL,**

manager\_id **int** **NOT** **NULL,**

emp\_id **int** **NOT** **NULL,**

**[timestamp]** **DATETIME** **NOT** **NULL,**

achievement **VARCHAR(**20**)** **CHECK** **([**Achievement**]=**'achieved' **OR** **[**Achievement**]=**'failed'**)**

**CONSTRAINT** **[**PK\_PREF\_APPRAISAL**]** **PRIMARY** **KEY** CLUSTERED **([**target\_code**]** **ASC,** **[**manager\_id**]** **ASC,** **[**emp\_id**]** **ASC,** **[Timestamp]** **ASC),**

**FOREIGN** **KEY** **(**emp\_id**)** **REFERENCES** Employee**(**emp\_id**),**

**FOREIGN** **KEY** **(**manager\_id**)** **REFERENCES** Employee**(**emp\_id**)**

**)**

**CREATE** **TABLE** Employee **(**

emp\_id **int** **IDENTITY(**1**,**1**)** **NOT** **NULL,**

emp\_FirstName **Varchar(**20**)** **NOT** **NULL,**

emp\_LastName **Varchar(**20**)** **NOT** **NULL,**

street **Varchar(**100**),**

city **Varchar(**30**),**

**[state]** **Varchar(**2**),**

zip\_Code **Varchar(**10**),**

phone **Varchar(**20**),**

**[**status**]** **Varchar(**2**),**

ss\_number **Varchar(**10**),**

**[**start\_date**]** **DATETIME** **NOT** **NULL** **DEFAULT** **getdate(),**

termination\_date **DATETIME,**

birth\_date **DATETIME** **NOT** **NULL,**

bene\_health\_ins **Char,**

bene\_life\_ins **Char,**

bene\_day\_care **Char,**

sex **Char** **CHECK(([**sex**]** **IS** **NULL)** **OR** **([**sex**]** **IN** **(**'M'**,** 'F'**))),**

salary **float** **NOT** **NULL** **CHECK([**salary**]** **>** 0**),**

phone\_work **Varchar(**20**),**

extension **Varchar(**6**),**

fax **Varchar(**20**),**

email **Varchar(**50**),**

cv **Varchar(**50**),**

manager\_id **int,**

branch\_code **int,**

pos\_numb **int** **NOT** **NULL,**

**PRIMARY** **KEY** **(**emp\_id**),**

**FOREIGN** **KEY** **(**manager\_id**)** **REFERENCES** Employee**(**emp\_id**),**

**FOREIGN** **KEY** **(**branch\_code**)** **REFERENCES** Branch**(**branch\_code**),**

**FOREIGN** **KEY** **(**pos\_numb**)** **REFERENCES** Position**(**pos\_numb**),**

**CONSTRAINT** **[**EMPLOYEE\_BIRTH\_DATE\_CONSTRAINT**]** **CHECK** **([**Birth\_Date**]<[**Start\_Date**]),**

**CONSTRAINT** **[**EMPLOYEE\_TERMINATION\_DATE\_CONSTRAINT**]** **CHECK** **(([**termination\_date**]** **IS** **NULL)** **OR** **([**Termination\_Date**]** **>** **[**Start\_Date**]))**

**)**

Deze constraints zorgen ervoor dat Termination Date null moet zijn, of na de start date moet plaatsvinden. Ook moet Birth Date voor de Start Date zijn

**CREATE** **TABLE** Department **(**

dept\_id **int** **IDENTITY(**1**,**1**) NOT** **NULL,**

dept\_name **Varchar(**255**)** **NOT** **NULL,**

dept\_head\_id **int** **NOT** **NULL,**

**PRIMARY** **KEY** **(**dept\_id**),**

**FOREIGN** **KEY** **(**dept\_head\_id**)** **REFERENCES** Employee**(**emp\_id**)**

**)**

**CREATE** **TABLE** Branch **(**

branch\_code **int** **IDENTITY(**1**,**1**) NOT** **NULL,**

address1 **Varchar(**50**)** **NOT** **NULL,**

address2 **Varchar(**50**),**

city **Varchar(**40**),**

region **Varchar(**50**),**

postal\_zone **Varchar(**10**),**

dept\_id **int** **NOT** **NULL,**

country\_code **int,**

**PRIMARY** **KEY** **(**branch\_code**),**

**FOREIGN** **KEY** **(**country\_code**)** **REFERENCES** Country**(**country\_code**),**

**FOREIGN** **KEY** **(**dept\_id**)** **REFERENCES** Department**(**dept\_id**)**

**)**

**CREATE** **TABLE** Product\_forecast **(**

product\_number **Int** **NOT** **NULL,**

**[year]** **smallInt** **NOT** **NULL,**

**[month]** **Tinyint** **NOT** **NULL,**

expected\_volume **Int** **CHECK(([**expected\_volume**]** **IS** **NULL)** **OR** **([**expected\_volume**]** **>** 0**)),**

**CONSTRAINT** **[**PK\_PRODUCT\_FORECAST**]** **PRIMARY** **KEY** CLUSTERED **([**product\_number**]** **ASC,** **[Year]** **ASC,** **[Month]** **ASC),**

**FOREIGN** **KEY** **(**product\_number**)** **REFERENCES** Product**(**product\_number**)**

**)**

**CREATE** **TABLE** Storage **(**

storage\_code **Int** **IDENTITY(**1**,**1**) NOT** **NULL,**

city **Varchar(**50**) NOT** **NULL,**

country\_code **int NOT** **NULL,**

**PRIMARY** **KEY** **(**storage\_code**),**

**FOREIGN** **KEY** **(**country\_code**)** **REFERENCES** Country**(**country\_code**)**

**)**

**CREATE** **TABLE** Campaign **(**

pr\_number **Int** **NOT** **NULL,**

product\_number **int** **NOT** **NULL,**

discount **float** **NOT** **NULL** **CHECK** **(([**discount**]** **>=** 0**)** **AND** **([**discount**]** **<=** 100**))**

**CONSTRAINT** **[**PK\_CAMPAIGN**]** **PRIMARY** **KEY** CLUSTERED **([**pr\_number**]** **ASC,** **[**product\_number**]** **ASC),**

**FOREIGN** **KEY** **(**product\_number**)** **REFERENCES** Product**(**product\_number**)**

**)**

**CREATE** **TABLE** Product **(**

product\_number **int** **IDENTITY(**1**,**1**)** **NOT** **NULL,**

introduction\_date **DATE** **NOT** **NULL,**

product\_type\_id **int** **NOT** **NULL,**

production\_cost **Float** **NOT** **NULL** **CHECK** **([**production\_cost**]** **>=** 0**),**

margin **Float** **NOT** **NULL** **CHECK** **([**margin**]** **>=** 0**),**

product\_image **Varchar(**255**)** **NOT** **NULL,**

product\_name **Varchar(**255**),**

product\_description **Varchar(**255**),**

product\_size **Varchar(**20**)** **CHECK([**product\_size**]** **IN** **(**'x-small'**,** 'small'**,** 'medium'**,** 'large'**,** 'x-large'**,** 'xx-large'**,** 'xxx-large'**)),**

product\_color **Varchar(**255**),**

**PRIMARY** **KEY** **(**product\_number**),**

**FOREIGN** **KEY** **(**product\_type\_id**)** **REFERENCES** Product\_type**(**product\_type\_id**)**

**)**

**CREATE** **TABLE** **[Order]** **(**

order\_id **int** **IDENTITY(**1**,**1**)** **NOT** **NULL,**

retailer\_site\_code **int** **NOT** **NULL,**

cust\_id **int** **NOT** **NULL,**

emp\_id **int** **NOT** **NULL,**

sales\_branch\_code **int** **NOT** **NULL,**

order\_date **DATETIME** **NOT** **NULL** **DEFAULT** **getdate(),**

order\_method\_code **int** **NOT** **NULL,**

finance\_code **int,**

region **Varchar(**50**),**

status **Varchar(**10**)** **NOT** **NULL** **CHECK([**status**]** **=** 'ongoing' **OR** **[**status**]** **=** 'done'**),**

cust\_discount **float** **NOT** **NULL** **DEFAULT** 0 **CHECK(([**cust\_discount**]** **>=** 0**)** **AND** **([**cust\_discount**]** **<=** 8**)),**

**PRIMARY** **KEY** **(**order\_id**),**

**FOREIGN** **KEY** **(**emp\_id**)** **REFERENCES** Employee**(**emp\_id**),**

**FOREIGN** **KEY** **(**cust\_id**)** **REFERENCES** Customer**(**id**),**

**FOREIGN** **KEY** **(**order\_method\_code**)** **REFERENCES** Order\_Method**(**order\_method\_code**),**

**FOREIGN** **KEY** **(**retailer\_site\_code**)** **REFERENCES** Retailer\_Site**(**retailer\_site\_code**),**

**FOREIGN** **KEY** **(**finance\_code**)** **REFERENCES** Order\_finance**(**finance\_code**),**

**CONSTRAINT** **[**ORDER\_ORDER\_DATE\_CONSTRAINT**]** **CHECK([**order\_date**]** **>=** **getdate())**

**)**

**CREATE** **TABLE** Customer **(**

id **int** **IDENTITY(**1**,**1**)** **NOT** **NULL,**

fname **Varchar(**255**),**

lname **Varchar(**255**),**

**[**address**]** **Varchar(**255**),**

city **Varchar(**40**),**

**[state]** **Varchar(**255**),**

zip **Varchar(**6**),**

phone **Varchar(**20**),**

company\_name **Varchar(**255**),**

**[type]** **Char** **NOT** **NULL,**

discount **float** **NOT** **NULL** **DEFAULT** 0 **CHECK(([**discount**]** **>=** 0**)** **AND** **([**discount**]** **<=** 8**)),**

max\_quantity\_order **int,**

**PRIMARY** **KEY** **(**id**),**

**CONSTRAINT** **[**CUSTOMER\_TYPE\_MAX\_QUANTITY\_ORDER**]** **CHECK** **((([type]** **=** 'B'**)** **AND** **([**max\_quantity\_order**]** **IS** **NULL)** **AND** **([**discount**]** **>=** 0**))** **OR** **(([type]** **=** 'S'**)** **AND** **([**discount**]** **=** 0**)** **AND** **([**max\_quantity\_order**]** **>** 0**)** **AND** **([**max\_quantity\_order**]** **IS** **NOT** **NULL)))**

**)**

Deze constraint zorgt ervoor dat als het type klant B is, de klant geen max quantity order heeft, en altijd een discount boven 0 heeft. Als het type klant S is, dan is de discount altijd 0, en heeft de klant altijd een max quantity order. Verder moet type altijd B of S zijn en kan het niets anders zijn.

**CREATE** **TABLE** Sales\_target **(**

emp\_id **int** **NOT** **NULL,**

sales\_year **SmallInt** **NOT** **NULL,**

sales\_period **int** **NOT** **NULL,**

retailer\_name **Varchar(**50**)** **NOT** **NULL,**

product\_number **int** **NOT** **NULL,**

sales\_target **Float** **NOT** **NULL** **CHECK([**sales\_target**]** **>** 0**),**

retailer\_code **int** **NOT** **NULL,**

**CONSTRAINT** **[**PK\_SALES\_TARGET**]** **PRIMARY** **KEY** CLUSTERED **([**emp\_id**]** **ASC,** **[**sales\_year**]** **ASC,** **[**sales\_period**]** **ASC),**

**FOREIGN** **KEY** **(**emp\_id**)** **REFERENCES** Employee**(**emp\_id**),**

**FOREIGN** **KEY** **(**product\_number**)** **REFERENCES** Product**(**product\_number**),**

**FOREIGN** **KEY** **(**retailer\_code**)** **REFERENCES** Retailer**(**retailer\_code**)**

**)**

**CREATE** **TABLE** Product\_type **(**

product\_type\_code **int** **IDENTITY(**1**,**1**) NOT** **NULL,**

product\_line\_code **int,**

product\_type\_en **Varchar(**80**)** **NOT** **NULL,**

**PRIMARY** **KEY** **(**product\_type\_code**),**

**FOREIGN** **KEY** **(**product\_line\_code**)** **REFERENCES** Product\_Line**(**product\_line\_code**)**

**)**

**CREATE** **TABLE** Order\_item **(**

order\_item\_id **int** **IDENTITY(**1**,**1**)** **NOT** **NULL**

order\_id **int** **NOT** **NULL,**

product\_number **int** **NOT** **NULL,**

quantity **SmallInt** **NOT** **NULL** **DEFAULT** 1 **CHECK([**quantity**]** **>** 0**),**

ship\_date **DATE** **NOT** **NULL,**

price\_each **FLOAT** **NOT** **NULL,**

**PRIMARY** **KEY** **(**order\_item\_id**),**

**FOREIGN** **KEY** **(**order\_id**)** **REFERENCES** **Order(**order\_id**),**

**FOREIGN** **KEY** **(**product\_number**)** **REFERENCES** Product**(**product\_number**)**

**)**

**CREATE** **TABLE** Returned\_item **(**

return\_code **int** **NOT** **NULL,**

return\_date **Datetime** **NOT** **NULL,**

order\_item\_id **int** **NOT** **NULL,**

return\_reason **int** **NOT** **NULL,**

return\_quantity **int** **NOT** **NULL,**

**PRIMARY** **KEY** **(**return\_code**),**

**FOREIGN** **KEY** **(**return\_reason**)** **REFERENCES** Return\_Reason**(**return\_reason\_code**),**

**FOREIGN** **KEY** **(**order\_item\_id**)** **REFERENCES** Order\_item**(**order\_item\_id**)**

**)**

**CREATE** **TABLE** Retailer **(**

retailer\_code **int** **IDENTITY(**1**,**1**)** **NOT** **NULL,**

company\_name **Varchar(**255**)** **NOT** **NULL,**

retailer\_type\_code **int** **NOT** **NULL,**

**PRIMARY** **KEY** **(**retailer\_code**),**

**FOREIGN** **KEY** **(**retailer\_type\_code**)** **REFERENCES** Retailer\_type**(**retailer\_type\_code**)**

**)**

**CREATE** **TABLE** Retailer\_site **(**

retailer\_site\_code **int** **IDENTITY(**1**,**1**)** **NOT** **NULL,**

retailer\_code **int** **NOT** **NULL,**

address1 **Varchar(**50**)** **NOT** **NULL,**

address2 **Varchar(**50**),**

city **Varchar(**40**),**

region **Varchar(**50**),**

postal\_zone **Varchar(**10**),**

country\_code **int NOT NULL,**

active\_indicator **bit NOT NULL,**

**PRIMARY** **KEY** **(**retailer\_site\_code**),**

**FOREIGN** **KEY** **(**country\_code**)** **REFERENCES** Country**(**country\_code**),**

**FOREIGN** **KEY** **(**retailer\_code**)** **REFERENCES** Retailer**(**retailer\_code**)**

**)**

**CREATE** **TABLE** Booking **(**

booking\_id **int** **IDENTITY(**1**,**1**)** **NOT** **NULL,**

transport\_to **Varchar(**255**),**

transport\_from **Varchar(**255**),**

vac\_id **int** **NOT** **NULL,**

**[**start\_date**]** **DATE** **NOT** **NULL,**

**PRIMARY** **KEY** **(**booking\_id**),**

**FOREIGN** **KEY** **(**vac\_id**)** **REFERENCES** Vacation**(**vac\_id**)**

**)**

**CREATE** **TABLE** Traveller **(**

customer\_id **int** **NOT** **NULL,**

booking\_id **int** **NOT** **NULL,**

iBan **Varchar(**255**),**

birth\_date **DATE** **NOT** **NULL,**

sex **Char** **NOT** **NULL,**

**CONSTRAINT** **[**PK\_TRAVELLER**]** **PRIMARY** **KEY** CLUSTERED **([**customer\_id**]** **ASC,** **[**booking\_id**]** **ASC),**

**FOREIGN** **KEY** **(**customer\_id**)** **REFERENCES** Customer**(**customer\_id**),**

**FOREIGN** **KEY** **(**booking\_id**)** **REFERENCES** Booking**(**booking\_id**)**

**)**

**CREATE** **TABLE** Accommodation **(**

acc\_id **int** **IDENTITY(**1**,**1**)** **NOT** **NULL,**

**[**address**]** **Varchar(**255**)** **NOT** **NULL,**

contact **Varchar(**255**),**

phone\_number **Varchar(**20**)** **NOT** **NULL,**

country\_code **int NOT** **NULL,**

**PRIMARY** **KEY** **(**acc\_id**),**

**FOREIGN** **KEY** **(**country\_code**)** **REFERENCES** Country**(**country\_code**)**

**)**

**CREATE** **TABLE** Acc\_vac **(**

acc\_id **int** **NOT** **NULL,**

vac\_id **int** **NOT** **NULL,**

**CONSTRAINT** **[**PK\_ACC\_VAC**]** **PRIMARY** **KEY** CLUSTERED **([**acc\_id**]** **ASC,** **[**vac\_id**]** **ASC),**

**FOREIGN** **KEY** **(**acc\_id**)** **REFERENCES** Accommodation**(**acc\_id**),**

**FOREIGN** **KEY** **(**vac\_id**)** **REFERENCES** Vacation**(**vac\_id**)**

**)**

**CREATE** **TABLE** Excursion **(**

ex\_id **int** **IDENTITY(**1**,**1**)** **NOT** **NULL,**

description **Varchar(**255**),**

price **float** **NOT** **NULL** **CHECK([**price**]** **>** 0**),**

guide\_name **Varchar(**50**)** **NOT** **NULL,**

guide\_phone **Varchar(**20**)** **NOT** **NULL,**

**PRIMARY** **KEY** **(**ex\_id**),**

**)**

**CREATE** **TABLE** Vac\_ex **(**

ex\_id **int** **NOT** **NULL,**

vac\_id **int** **NOT** **NULL,**

**CONSTRAINT** **[**PK\_VAC\_EX**]** **PRIMARY** **KEY** CLUSTERED **([**ex\_id**]** **ASC,** **[**vac\_id**]** **ASC),**

**FOREIGN** **KEY** **(**ex\_id**)** **REFERENCES** Excursion**(**ex\_id**),**

**FOREIGN** **KEY** **(**vac\_id**)** **REFERENCES** Vacation**(**vac\_id**)**

**)**

**CREATE** **TABLE** Inventory **(**

inventory\_year **SmallInt** **NOT** **NULL,**

inventory\_month **TinyInt** **NOT** **NULL,**

inventory\_count **int** **NOT** **NULL,**

storage\_code **int** **NOT** **NULL,**

product\_number **int** **NOT** **NULL,**

**CONSTRAINT** **[**PK\_INVENTORY**]** **PRIMARY** **KEY** CLUSTERED **([**inventory\_year**]** **ASC,** **[**inventory\_month**]** **ASC,** **[**storage\_code**]** **ASC,** **[**product\_number**]** **ASC),**

**FOREIGN** **KEY** **(**product\_number**)** **REFERENCES** Product**(**product\_number**),**

**FOREIGN** **KEY** **(**storage\_code**)** **REFERENCES** Storage**(**storage\_code**)**

**)**

Procedures

**GetProductSellPrice**

**CREATE** **PROCEDURE** getProductSellPrice

@product\_number **INT,**

@sellprice **FLOAT** **OUT**

**AS**

**BEGIN**

**SELECT** @discount**=**Discount **FROM** Campaign **JOIN** Promotion **ON** Campaign**.**Pr\_Number **=** Promotion**.**Pr\_Number **WHERE** Promotion**.**Date\_Start **<=** **getdate()** **AND** Promotion**.**Date\_End **>** **getdate()** **AND** Campaign**.**Product\_Number**=**@product\_number

**SELECT** @sellprice**=(**production\_cost **+** margin**)** **FROM** Product **WHERE** product\_number**=**@product\_number

**IF** @discount **IS** **NULL**

**SET** @discount **=** 0

**SET** @sellprice **-=** @sellprice **\*** **(**@discount **/** 100**)**

**END**

Deze procedure berekent de verkoopprijs van een opgegeven product door de productie kosten plus de margin terug te geven. Hier wordt ook rekening gehouden met de korting op een product in de huidige periode.

**CREATE** **PROCEDURE** getTotalOrderPrice**(**

-- Add the parameters for the stored procedure here

@order\_id **INT,**

@calc\_discount **BIT** **=** 0**,**

@totalprice **FLOAT** **OUT**

**)**

**AS**

**DECLARE** @quantity **INT**

**DECLARE** @price\_each **FLOAT**

**SET** @totalprice **=** 0

**DECLARE** the\_cursor **CURSOR** FAST\_FORWARD

**FOR** **SELECT** quantity**,** price\_each **FROM** Order\_item **WHERE** order\_id**=**@order\_id

**OPEN** the\_cursor

**FETCH** **NEXT** **FROM** the\_cursor **INTO** @quantity**,** @price\_each

**WHILE** @@FETCH\_STATUS **=** 0

**BEGIN**

**SET** @totalprice **+=** **(**@price\_each **\*** @quantity**)**

**FETCH** **NEXT** **FROM** the\_cursor **INTO** @quantity**,** @price\_each

**END**

**CLOSE** the\_cursor

**DEALLOCATE** the\_cursor

**IF** **(**@calc\_discount **=** 1**)**

**BEGIN**

**DECLARE** @discount **FLOAT**

**SELECT** @discount**=**cust\_discount **FROM** **[order]** **WHERE** order\_id**=**@order\_id

**SET** @totalprice **-=** @totalprice **\*** **(**@discount**/**100**)**

**END**

Deze procedure berekent de totaalprijs van de opgegeven order. Dit wordt gedaan door van elk item in de order de eerste procedure aan te roepen en de prijs die wordt terug gegeven te vermenigvuldigen met de hoeveelheid die er van het product besteld is. Nadat dit is gedaan wordt de persoonlijke korting van de customer nog toegepast.

**CREATE** **PROCEDURE** getTotalVacationPrice**(**

@vac\_id **INT,**

@price **FLOAT** **OUT**

**)**

**AS**

**BEGIN**

**SELECT** @price**=**price **FROM** vacation **JOIN** Booking **ON** Booking**.**Vac\_Id **=** Vacation**.**Vac\_Id **WHERE** Booking**.**Booking\_Id**=**@booking\_id

**SELECT** @price**+=ISNULL(SUM(**price**),** 0**)** **FROM** Excursion **JOIN** Bo\_ex **AS** Koppel **ON** Koppel**.**ex\_id **=** Excursion**.**Ex\_Id **WHERE** koppel**.**booking\_id**=**@booking\_id

**END**

Deze procedure berekent de prijs van een gegeven boeking. Dit is de prijs van de bijbehorende vakantie plus de prijs van alle excursies die bij de boeking horen. Een vakantie zonder excursies werkt ook.

**CREATE** **PROCEDURE** getTotalSellPriceFromEmployee**(**

@emp\_id **INT,**

@total\_order\_price **FLOAT** **OUT**

**)**

**AS**

**DECLARE** @current\_order\_id **INT**

**SET** @total\_order\_price **=** 0

**DECLARE** the\_cursor **CURSOR** FAST\_FORWARD

**FOR** **SELECT** order\_id **FROM** **[Order]** **WHERE** emp\_id**=**@emp\_id

**OPEN** the\_cursor

**FETCH** **NEXT** **FROM** the\_cursor **INTO** @current\_order\_id

**WHILE** @@FETCH\_STATUS **=** 0

**BEGIN**

**DECLARE** @orderprice **FLOAT**

**EXEC** getTotalOrderPrice @order\_id **=** @current\_order\_id**,** @totalprice **=** @orderprice **OUTPUT**

**SET** @total\_order\_price **+=** @orderprice

**FETCH** **NEXT** **FROM** the\_cursor **INTO** @current\_order\_id

**END**

**CLOSE** the\_cursor

**DEALLOCATE** the\_cursor

Deze procedure berekend de totale prijs van alle orders die bij een employee horen.

Triggers  
**CREATE** **TRIGGER** OutdoorParadise**.**SalaryCheck

**ON** OutdoorParadise**.**Employee

**AFTER** **INSERT,UPDATE**

**AS**

**DECLARE** @emp **FLOAT**

**DECLARE** @man **FLOAT**

**select** @man **=** salary **FROM** inserted**;**

**select** @emp **=** **MAX(**salary**)** **FROM** employee

**WHERE** manager\_id **=** **(select** emp\_id **from** inserted**);**

**IF** @man **<=** @emp

**BEGIN**

-- SET NOCOUNT ON added to prevent extra result sets from

-- interfering with SELECT statements.

**SET** NOCOUNT **ON;**

-- Insert statements for trigger here

RAISERROR**(**'The employee is not allowed to be payed more than their manager.'**,** 16**,** 1**);**

**ROLLBACK;**

**RETURN**

**END**

**GO**

Controleert of het salaris van de manager niet lager is dan de ondergeschikten van hem die het meest verdient. Als dit wel zo is verschijnt er een error, en wordt de verandering die voor deze situatie zorgde ongedaan gemaakt.

**CREATE** **TRIGGER** dbo**.**SalaryCheck

**ON** dbo**.**Employee

**AFTER** **INSERT,UPDATE**

**AS**

**DECLARE** @emp **FLOAT**

**DECLARE** @man **FLOAT**

**select** @emp **=** inserted**.**salary **FROM** inserted**;**

**select** @man **=** salary **FROM** employee

**WHERE** emp\_id **=** **(select** manager\_id **from** inserted**);**

**IF** @man **<=** @emp

**BEGIN**

-- SET NOCOUNT ON added to prevent extra result sets from

-- interfering with SELECT statements.

**SET** NOCOUNT **ON;**

-- Insert statements for trigger here

RAISERROR**(**'The employee is not allowed to be payed more than their manager.'**,** 16**,** 1**);**

**ROLLBACK;**

**RETURN**

**END**

**GO**

Zorgt ervoor dat het salaris van een werknemer wordt gecontroleerd op het moment dat deze wordt toegevoegd/aangepast. Dit wordt gedaan om te voorkomen dat een werknemer meer kan verdienen dan zijn manager. Als dit wel zo is verschijnt er een error, en wordt de verandering die voor deze situatie zorgde ongedaan gemaakt. Dit was een eis van de opdrachtgever.

**CREATE** **TRIGGER** **[**dbo**].[**Default\_Price\_Each**]**

**ON** **[**dbo**].[**Order\_item**]**

**AFTER** **INSERT**

**AS**

**BEGIN**

**DECLARE** @productnumber **INT**

**SELECT** @productnumber**=**product\_id **FROM** inserted

**DECLARE** @defaultprice **FLOAT**

**EXEC** getProductSellPrice @product\_number **=** @productnumber**,** @sellprice **=** @defaultprice **OUTPUT**

**UPDATE** Order\_item **SET** price\_each**=**@defaultprice **FROM** inserted **WHERE** Order\_Item**.**order\_item\_id **=** inserted**.**order\_item\_id

**END**

Deze trigger zorgt ervoor dat elke keer wanneer er een item aan een order wordt toegevoegd, de huidige prijs van het item inclusief enige bonussen

**CREATE** **TRIGGER** dbo**.**ManagersCheck

**ON** dbo**.**Employee

**AFTER** **INSERT,** **UPDATE**

**AS**

**BEGIN**

-- SET NOCOUNT ON added to prevent extra result sets from

-- interfering with SELECT statements.

**SET** NOCOUNT **ON;**

**DECLARE** @manager\_id **INT**

**SELECT** @manager\_id **=** Manager\_Id **FROM** inserted

**IF** @manager\_id **IS** **NOT** **NULL**

**BEGIN**

**DECLARE** @keer **INT**

**SELECT** @keer **=** **COUNT(**emp\_id**)** **FROM** Employee **WHERE** manager\_id **=** @manager\_id**;**

**IF** @keer **>** 12

**BEGIN**

RAISERROR**(**'The manager is already assigned to 12 employees and cannot be assigned to more.'**,** 16**,** 1**);**

**ROLLBACK;**

**RETURN**

**END**

**END**

**END;**

Deze trigger zorgt ervoor dat een manager niet meer dan 12 employees kan hebben. Als dit wel zo is verschijnt er een error, en wordt de verandering die voor deze situatie zorgde ongedaan gemaakt. Dit was een eis van de opdrachtgever.

**CREATE** **TRIGGER** dbo**.**MinimumLoon

**ON** dbo**.**Employee

**AFTER** **INSERT,** **UPDATE**

**AS**

**BEGIN**

-- SET NOCOUNT ON added to prevent extra result sets from

-- interfering with SELECT statements.

**SET** NOCOUNT **ON;**

-- Insert statements for trigger here

**DECLARE** @salary **FLOAT**

**DECLARE** @branch\_code **INT**

**SELECT** @salary**=**salary**,** @branch\_code**=**Branch\_Code **FROM** inserted

**IF** @branch\_code **IS** **NOT** **NULL** **AND** @salary **IS** **NOT** **NULL**

**BEGIN**

**DECLARE** @minimum **FLOAT**

**SELECT** @minimum **=** Country**.**Minimum\_Wage

**FROM** Country **JOIN** Branch **ON** Country**.**Country\_code **=** Branch**.**Country\_code

**WHERE** Branch**.**branch\_code **=** @branch\_code

**IF** @salary **<** @minimum

**BEGIN**

RAISERROR**(**'The wage of the employee is less than the minimum wage of the country the employee is working in.'**,** 16**,** 1**);**

**ROLLBACK;**

**RETURN**

**END**

**END**

**END**

Deze trigger zorgt ervoor dat het salaris van een employee niet minder kan zijn dan het minimum salaris van het land van de branch waar de employee in werkt. Als dit wel zo is verschijnt er een error, en wordt de verandering die voor deze situatie zorgde ongedaan gemaakt.

**Security Outdoor Paradise**

Bij databases is het van groot belang dat de opgeslagen gegevens goed beveiligd zijn. Het is belangrijk dat de gegevens alleen toegankelijk zijn voor de mensen die rechten hebben om met die gegevens te mogen werken. We gaan ervoor zorgen dat de verschillende medewerkers van Outdoor Paradise, die bij de verschillende afdelingen werken, hun eigen rollen krijgen, waarmee ze beperkte acties kunnen uitvoeren op de database.

De afdeling HRM heeft toegang tot de tabellen die te maken hebben met de employees en courses, zoals Employee, Perf\_Appraisel en Training.  
De afdeling Sales heeft toegang tot de tabellen die te maken hebben met producten en verschillende reizen, zoals Product, Sales\_Target en Booking.  
De afdeling Marketing heeft toegang tot de tabellen die te maken hebben met de producten en de promotie van die producten, zoals Product, Promotion en Campaign.

Hieronder bevinden zich SQL-code van de rechten die de verschillende rollen(HRM\_Hans, HRM\_Frank, HRM\_Petra, Sales\_Bob, Sales\_Mark, Sales\_Cindy, Markt\_Sandra, Markt\_Bert, Markt\_Lisa) van de verschillende afdelingen(HRM, Sales, Marketing):

CREATE USER:  
CREATE USER HRM\_Hans;  
CREATE USER HRM\_Frank;  
CREATE USER HRM\_Petra;  
CREATE USER Sales\_Bob;  
CREATE USER Sales\_Mark;  
CREATE USER Sales\_Cindy;  
CREATE USER Markt\_Sandra;  
CREATE USER Markt\_Bert;  
CREATE USER Markt\_Lisa;

CREATE ROLE:  
CREATE ROLE HRM;  
CREATE ROLE Sales;  
CREATE ROLE Marketing;

GRANT:  
GRANT SELECT, INSERT, UPDATE  
ON OutdoorParadise.Employee, OutdoorParadise.Perf\_appraisel, OutdoorParadise.Target, OutdoorParadise.Training, OutdoorParadise.Emp\_Training, OutdoorParadise.Bonus, OutdoorParadise.Position  
TO HRM;

GRANT SELECT, INSERT, UPDATE  
ON OutdoorParadise.Product, OutdoorParadise.Product\_Forecast, OutdoorParadise.Product\_Type, OutdoorParadise.Product\_Line, OutdoorParadise.Sales\_Target, OutdoorParadise.Vacation, OudoorParadise.Excursion, OutdoorParadise.Booking, OutdoorParadise.Accommodation, OutdoorParadise.Traveller  
TO Sales;

GRANT SELECT, INSERT, UPDATE  
ON OutdoorParadise.Product, OutdoorParadise.Promotion, OutdoorParadise.Campaign  
TO Marketing;

GRANT HRM  
TO HRM\_Hans, HRM\_Frank, HRM\_Petra;

GRANT Sales  
TO Sales\_Bob, Sales\_Mark, Sales\_Cindy;

GRANT Marketing  
TO Markt\_Sandra, Markt\_Bert, Markt\_Lisa;

# Fysiek ontwerp

We hebben ervoor gekozen om 2 secundaire indexen aan te maken op de tabel Employee. Deze tabel hebben we gekozen omdat we van mening zijn dat hij niet vaak worden aangepast en door veel query’s gebruikt wordt.

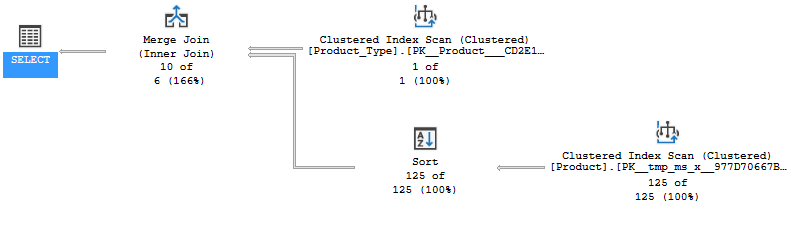
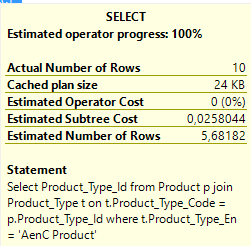
## Product

De tabel product is een tabel waarvan heel vaak data opgevraagd wordt. Ook heeft de tabel nu 125 records. We hebben daarom besloten een secundaire index toe te voegen op de kolommen Product\_Type en Manager\_Id. Deze kolommen hebben we gekozen aangezien product type iets is wat mensen vaak opzoeken. En Manager\_Id aangezien er niet vaak managers bij komen en wij verwachten dat er veel query’s zijn die van deze index profiteren.

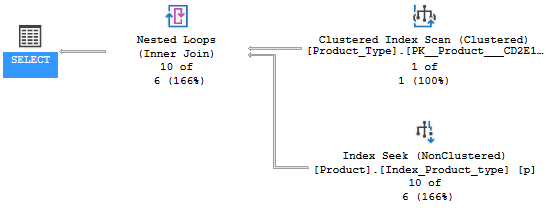
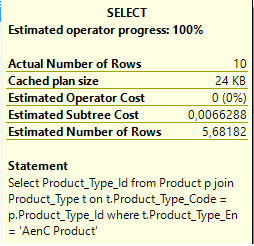
### Preformance test Product\_Type

C:\Users\Cas Ros\Documents\HHS\jaar 2\periode 3\Project\Project-jaar-2-periode-3\fysiek ontwerp afb\productItemIndex1.PNG

We hebben de performance van de index getest met behulp van de bovenstaande query. Als resultaat kwam het onderstaande execution plan.

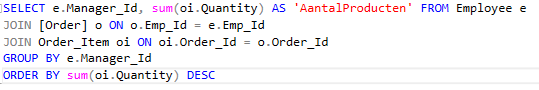


Uit het execution plan is niet veel informatie te herleiden aangezien de costen 100% of 166% zijn.

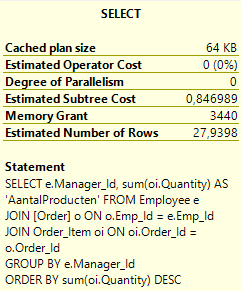
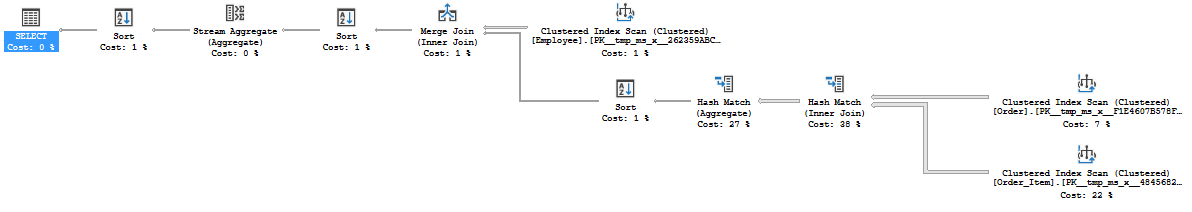


door de executies van deze query voor en na het aanmaken van de index te vergelijken zien we dat de subtree cost aanzienlijk is vermindert. Ook kan je aan het execution plan zien dat hij gebruik heeft gemaakt van de aangemaakte index. Het probleem met de kosten van 100% en 166% is er nog steeds, maar nu is er 1 stap minder in de execution planner.

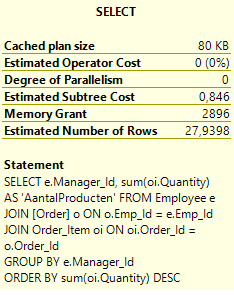
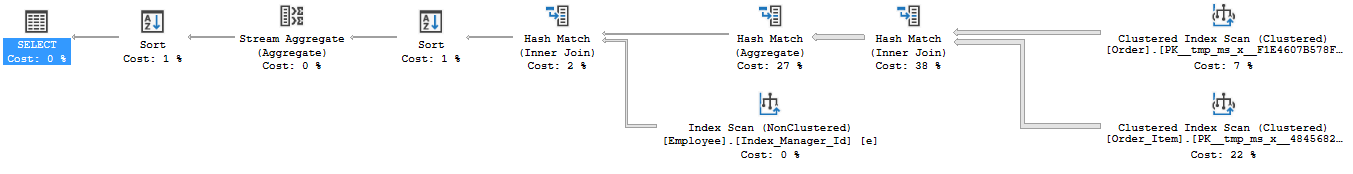
### Preformance test Manager\_Id



We hebben de performance van de index getest met behulp van de bovenstaande query. Als resultaat kwam het onderstaande execution plan.



Hierin is te zien dat de execution planner maar 1% nodig heeft voor de employee tabel. Toch hebben we ervoor gekozen hier de index op te doen aangezien de Order en Order\_Item tabellen zeer vaak met nieuwe data gevuld worden, waardoor indexen op deze tabellen constant bij gewerkt zouden moeten worden.



Ook bij dit execution plan is te zien dat na het aanmaken van de index de query hiervan gebruik maakt. Als resultaat heeft de query nu minder memory nodig. Ook is de cost die eerst 1% was nu 0%.

# Test rapportage

We hebben verschillende testen uitgevoerd op de onderdelen van de nieuwe database. Op de hele database hebben we een performance test geschreven om te kijken of er nog wijzigingen moet plaats vinden binnen de database structuur. Daarnaast heb we de verschillende database rollen getest om te kijken of de users daadwerkelijk de juiste rechten hebben en overigens hebben we 3 constrains, 3 triggers en 3 stored procedures getest. Hieronder bevinden zich de verschillende testgevallen.

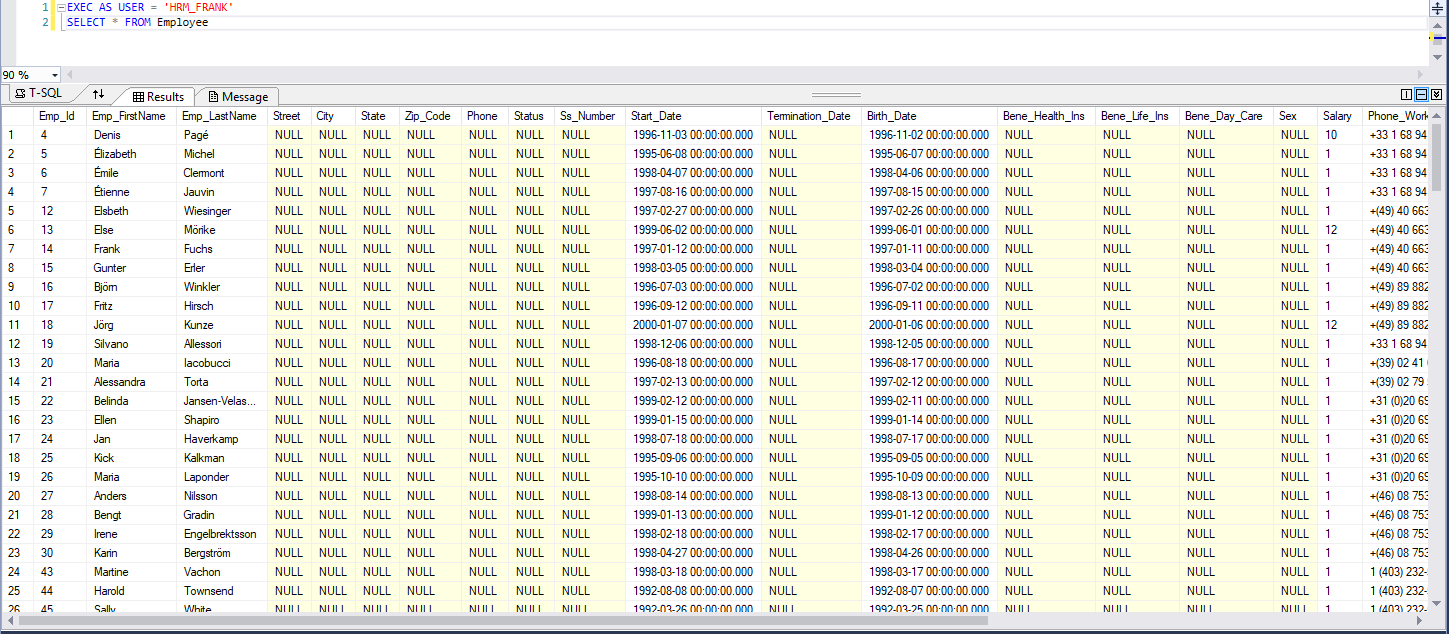


**Security (User en Role) testen**

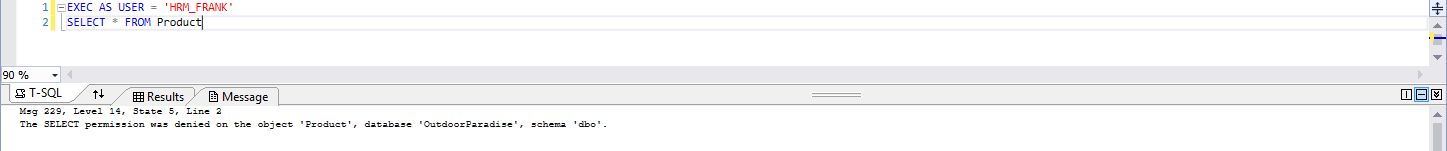
Bij de roles en users is getest of de toegekende permissions ook correct werkte. Hieronder staan een aantal afbeeldingen van de testen.

Bij afbeelding 1 en 2 is de Select test te zien. User HRM\_Frank heeft toegang tot selects in tabel Employee (afb. 1), maar niet in tabel product (afb. 2)

*Afbeelding 1: HRM Select test 1*

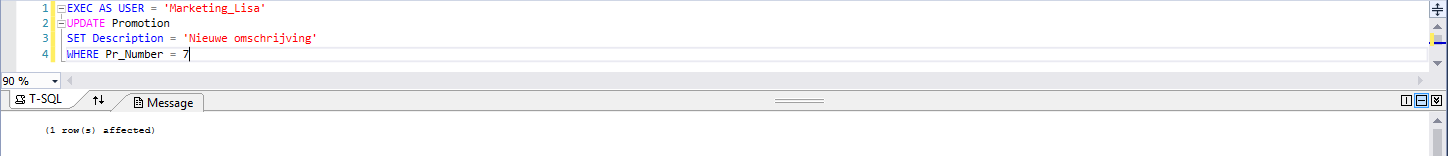


*Afbeelding 2: HRM Select test 2*

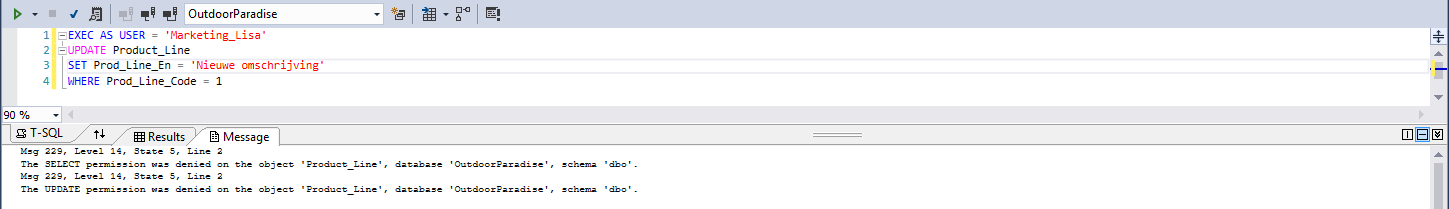


Bij afbeelding 3 en 4 is de Update test te zien. User Marketing\_Lisa mag wel updates uitvoeren op de tabel promotion (afb. 3), maar niet op de tabel product\_line (afb. 4)

*Afbeelding 3: Marketing Update test 1*

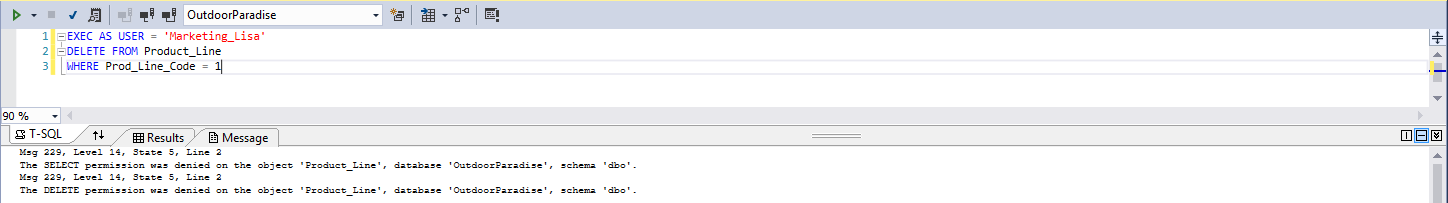


*Afbeelding 4: Marketing Update test 2*



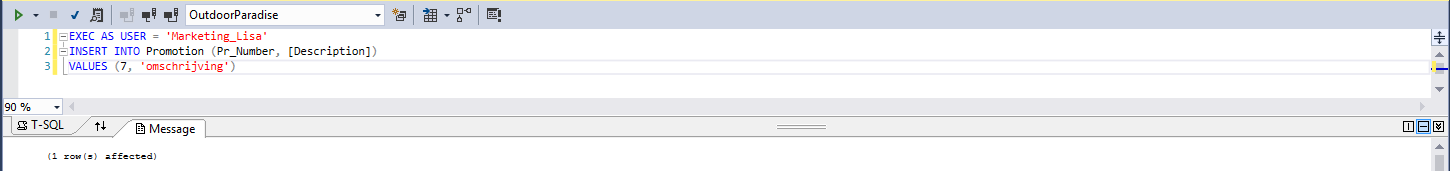
Bij afbeelding 5 is de delete test te zien. Geen van de users heeft het recht op het uitvoeren van een delete (afb. 5).

*Afbeelding 5: Marketing Delete test*

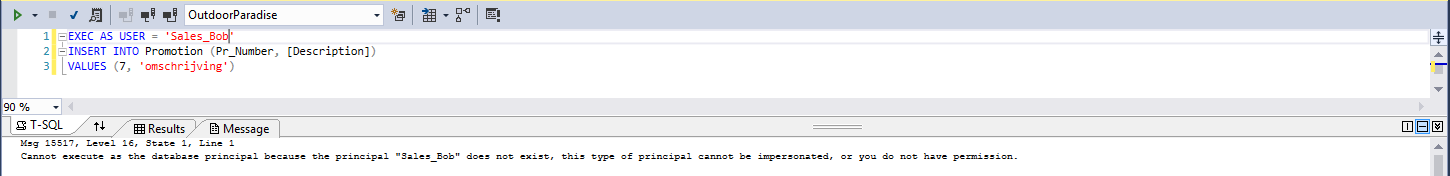


Bij afbeelding 6 en 7 is de insert getest. Marketing\_Lisa heeft wel toegang tot een insert op tabel promotion (afb. 6), terwijl Sales\_Bob geen toegang heeft tot een insert op tabel promotion (afb. 7)

*Afbeelding 6 : Marketing Insert test*



*Afbeelding 7: Sales Insert test*



Procedure Tests

**GetProductSellPrice**

Data in Database:



Product ID 4, Production\_Cost 10, Margin 3.

Verwachte verkoopprijs zonder enige bonus (karting): 13

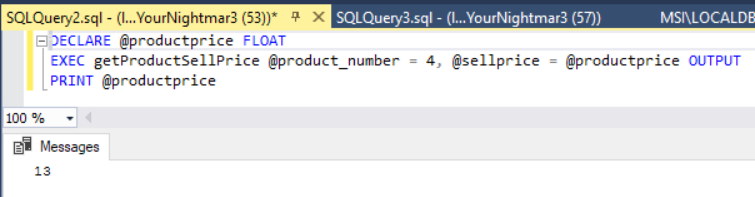
Procedure uitgevoerd zonder korting, maar wel met een historische korting, dus een korting op het product waarvan de datum voorbij is:

Korting:

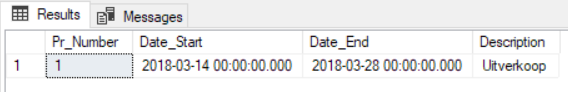


Verwachte uitvoer: 13

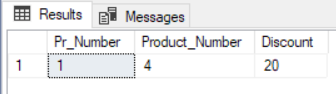
Werkelijke uitvoer:



Dan een test waarbij het einde van de korting in de toekomst ligt, wat dus betekend dat het product op dit moment korting heeft:

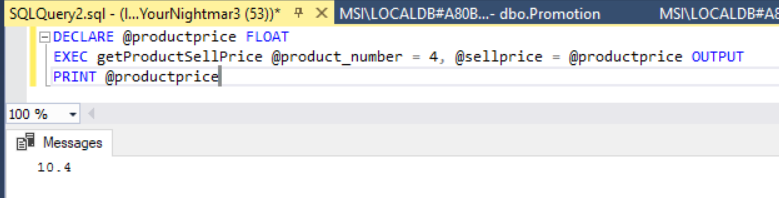


De korting bedraagt 20%:



Verwachte uitvoer: 13\*0.8=10.4

Werkelijke uitvoer:

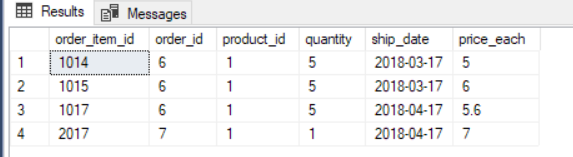


**GetTotalOrderPrice**

Er staat een order in de database:



Bij deze order horen drie items:



De totale prijs van deze order exclusief korting is:

(5\*5)+(5\*6)+(5\*5.6)= 83

De customer had op het moment van het plaatsen van deze order een persoonlijke korting van 5%.

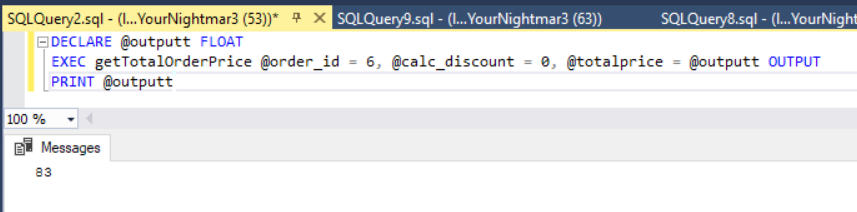
De totale prijs van deze order inclusief korting is:

83\*0.95=78.85

Procedure zonder het berekenen van de korting:

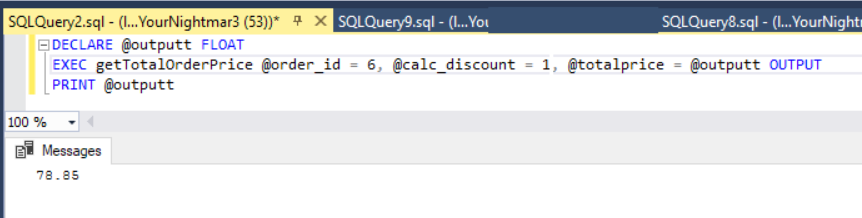
Verwachte uitvoer: 83

Werkelijke uitvoer:



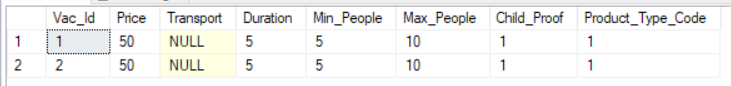
Procedure met het berekenen van de korting:

Verwachte uitvoer: 78.85

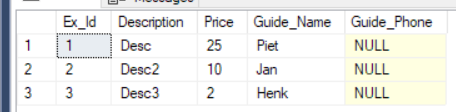
Werkelijke uitvoer: 

**GetTotalBookingPrice**

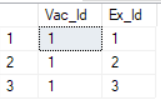
Er zijn twee vakanties in de database:



Ook zijn er drie excursies:

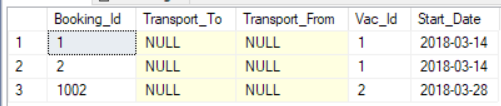


Vakantie ID 1 heeft alle drie de excusies:



Vakantie ID 2 heeft geen excursies.

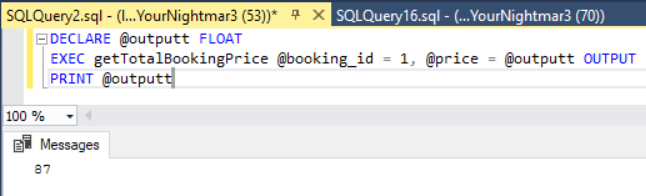
Verder zijn er drie boekingen in de database. Twee boekingen, ID 1 en 2, voor vakantie 1. Een boeking, ID 1002, voor vakantie 2.



De procedure neemt een boeking ID aan en berekent dan de prijs inclusief excursies.

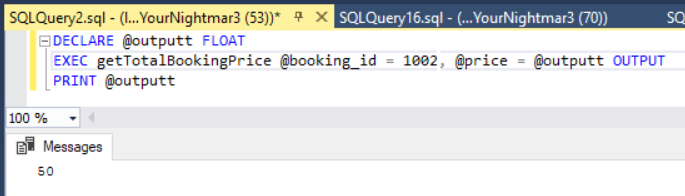
Verwachte uitvoer voor boeking ID 1: 50+25+10+2=87

Werkelijke uitvoer:



Verwachte uitvoer voor boeking ID 1002: 50

Werkelijke uitvoer:



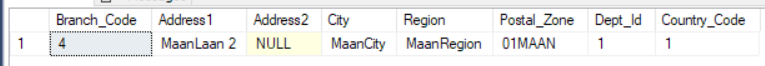
Trigger Tests

**MinimumLoon**

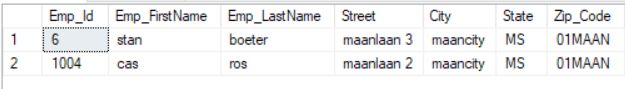
De database heeft een country met een minimumloon van 5.5.



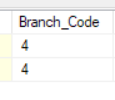
Verder is er een branch in MaanLand:



Er zijn twee employees:

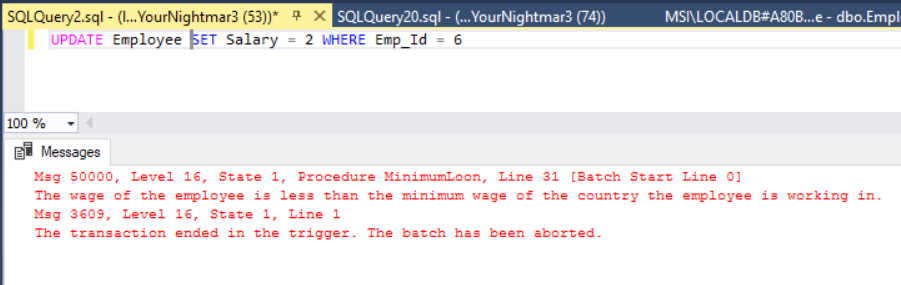


Die allebei in branch code 4 werken, en dus in Country\_Code 1:



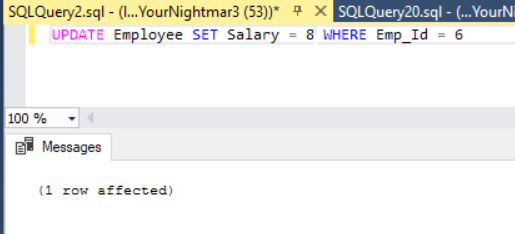
Dit betekent dat het minimumloon van beide employees 5.5 is.

Als we het loon van een employee proberen te veranderen naar iets lager dan 5.5:



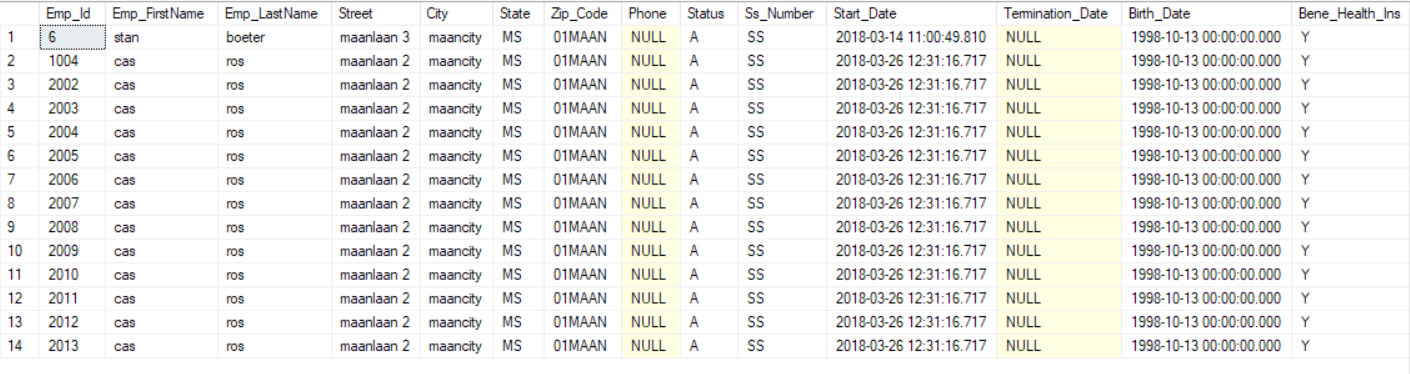
Dan kan dat niet door de trigger.

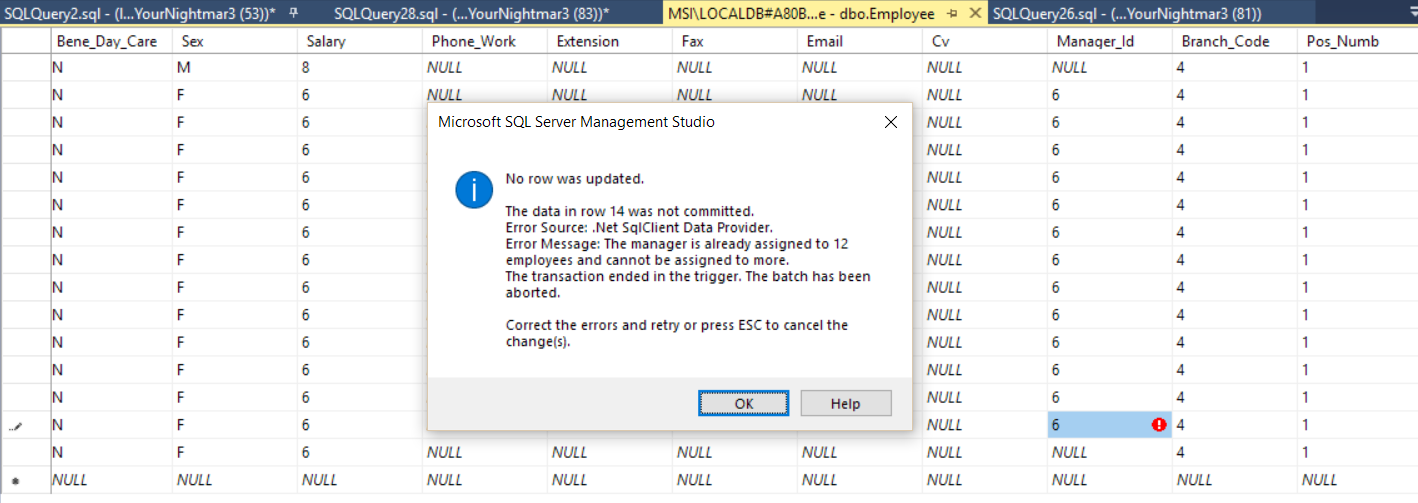
Veranderen naar iets hoger dan het minimumloon kan wel:



**ManagersCheck**

We hebben veertien employees in de database. Hiervan zijn dertien precies het zelfde, maar dat maakt voor de test niets uit. We gaan employee ID 6 als manager gebruiken bij de rest van de employees. Dit zijn er meer dan 12, dus als ik alle employees de manager 6 toewijs, zou de trigger dit proces moeten stoppen.

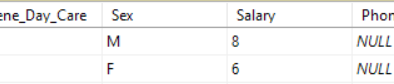


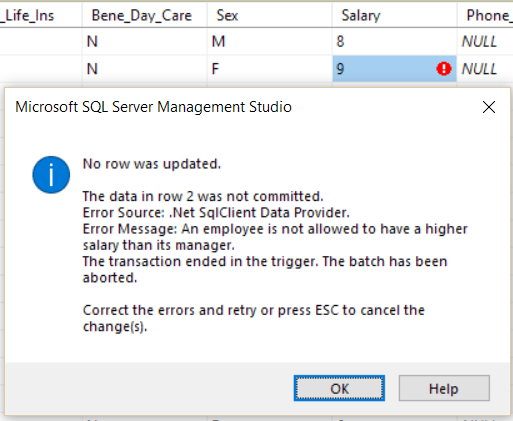


En dit werkt. Op update en insert.

**SalaryCheck**

Er zijn twee employees. De ene employee is de manager van de andere. De manager heeft een salaris van 8. De employee heeft een salaris van 6. Als we het salaris van de employee hoger proberen te maken dan het salaris van zijn manager, zou de trigger dit moeten stoppen.



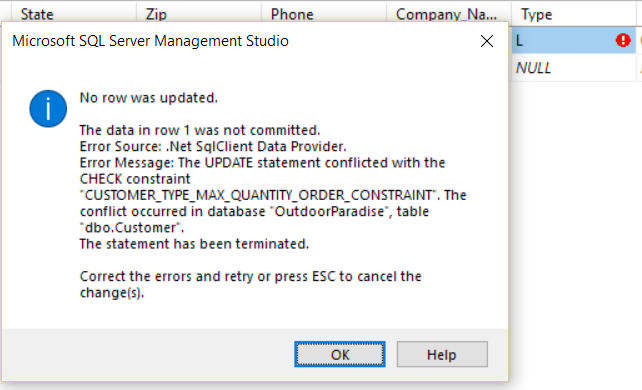


En dit klopt inderdaad. Een werknemer kan nooit een salaris hoger hebben dan zijn manager.

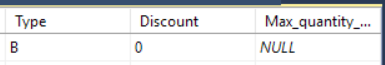
Constraint Tests

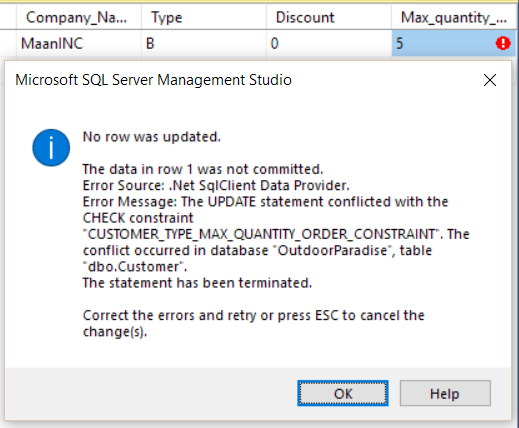
**Constraint CUSTOMER\_TYPE\_MAX\_QUANTITY\_ORDER op de tabel customer.**

Dit is een ingewikkelde constraint, maar is goed te testen. Ten eerste zou dit er voor moeten zorgen dat als we het veld “type” bij een customer veranderen naar iets anders dan een B of een S, dit niet mag.

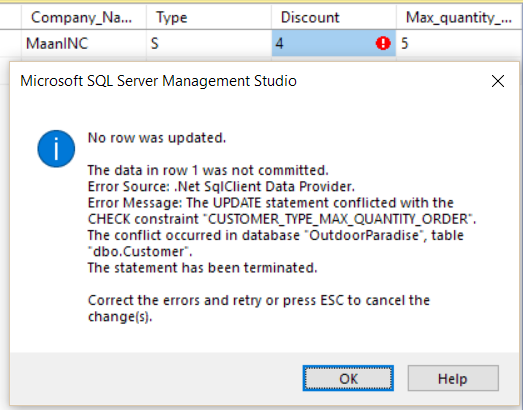


En dat klopt. Verder zou het constraint er voor moeten zorgen dat als type “B” is, max\_quantity\_order NULL moet zijn, en discount hoger of gelijk aan 0 moet zijn.

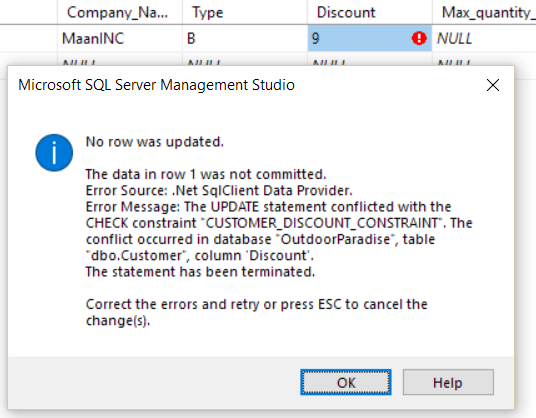




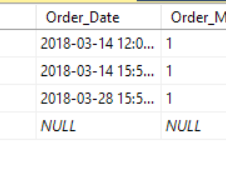
Ook dit klopt volledig. Verder als type “S” is, mag discount alleen 0 zijn, en moet max\_quantity\_order altijd meer dan 0 zijn.



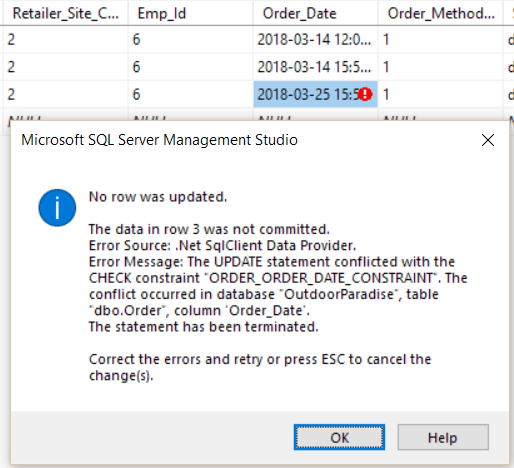
Verder mag Discount sowieso nooit meer dan 8 zijn, omdat dat weer een aparte check is.



**Constraint ORDER\_ORDER\_DATE\_CONSTRAINT op de tabel order**

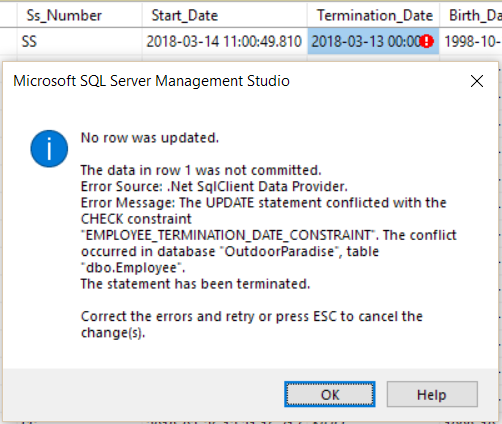


Als we een order proberen aan te maken met een datum in het verleden, dan kan dit niet. De datum op het moment van het schrijven van deze zin is 2018-03-27

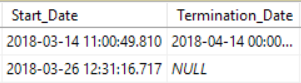


**Constraint EMPLOYEE\_TERMINATION\_DATE\_CONSTRAINT op de tabel Employee**

Een employee heeft een start\_date en een termination date. Deze constraint zorgt er voor dat de termination\_date niet voor de start date plaats kan vinden:



Een termination date na de start date mag wel. Ook mag termination date NULL zijn:



Performance Test

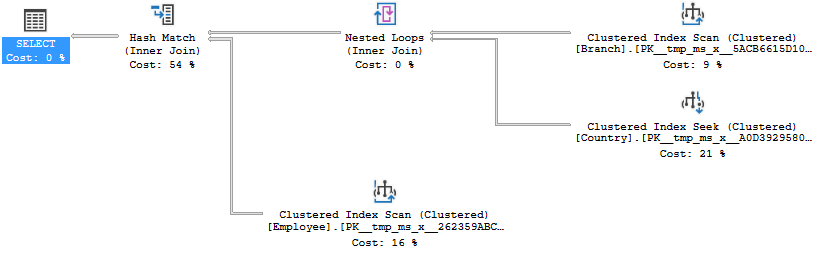
Hieronder bevinden zich 3 query’s die uitgevoerd zijn op de database. We hebben de queryexecutieplannen erbij gezet en bestudeerd. Hiermee hebben we conclusies kunnen maken of er enkele tabellen samengevoegd of gesplist moesten worden.

Query 1:



Hier mee kijken we welke medewerker in welk land werkt.

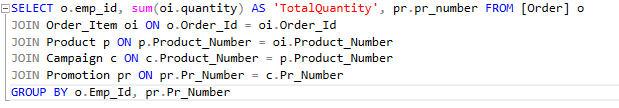
Queryexecutieplannen 1:



Conclusie:

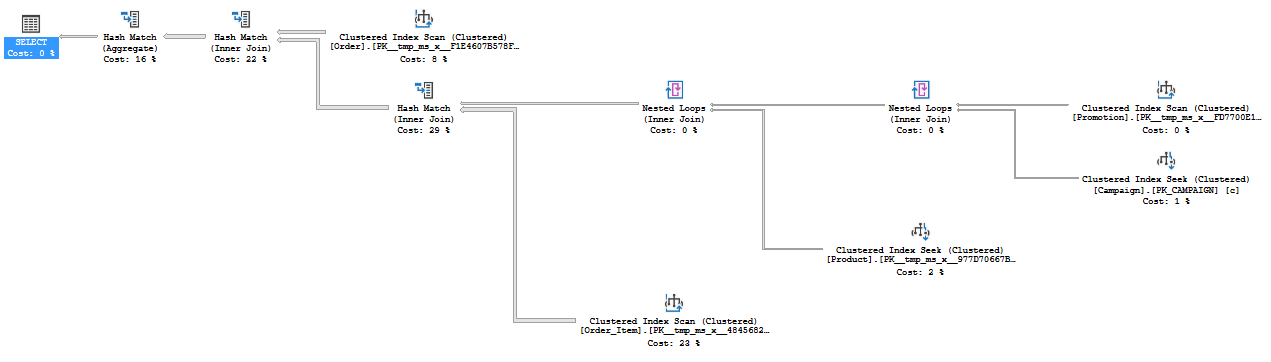
We kunnen concluderen dat het niet nog is om de verschillende tabellen te samenvoegen of te splitsen omdat we er van overtuigd zijn dat de attributen zich op de juiste plekken bevinden. Het kost ook niet veel tijd om de query uit te voeren.

Query 2:



Hier mee kijken we wat de totale hoeveelheid is van een product die door een medewerker besteld is.

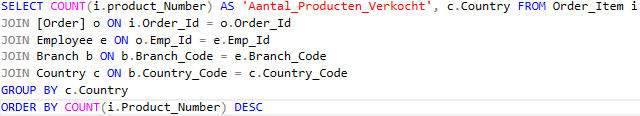
Queryexecutieplannen 2:



Conclusie:

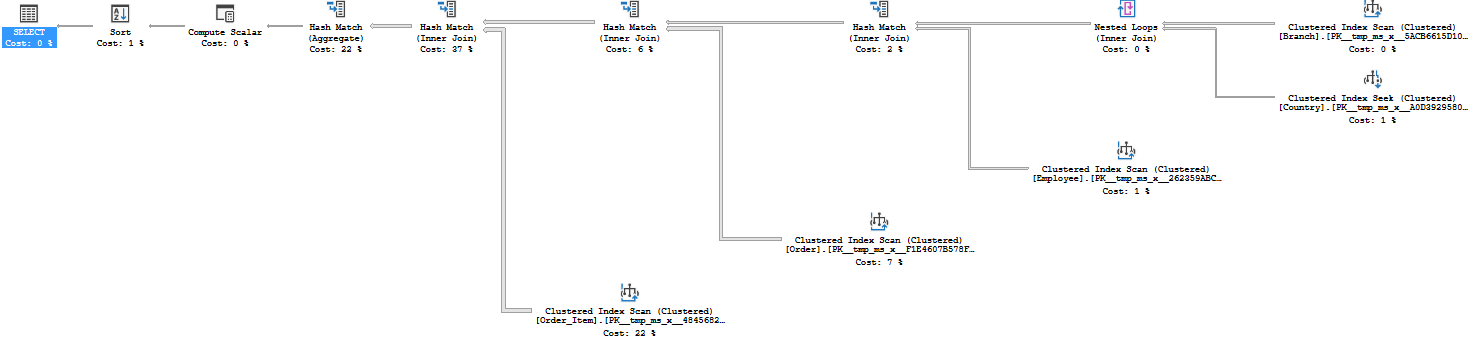
We kunnen concluderen dat het niet nog is om de verschillende tabellen te samenvoegen of te splitsen omdat we er van overtuigd zijn dat de attributen zich op de juiste plekken bevinden. Het kost wel wat meer tijd om de query uit te voeren, maar niet zoveel om er verandering in te brengen.

Query 3:



Hier mee kijken we hoeveel producten per land er besteld zijn.

Queryexecutieplannen 3:



Conclusie:

We kunnen concluderen dat het niet nog is om de verschillende tabellen samen te voegen of te splitsen omdat we er van overtuigd zijn dat de attributen zich op de juiste plekken bevinden. Er worden wel veel tabellen met elkaar gejoind om tot het juiste resultaat te komen, maar de joins zelfs kosten niet zoveel tijd. Het kost ook niet te veel tijd om de rest van de query uit te voeren.

## Conclusie

Na alle testen te hebben doorlopen hebben we besloten om de secundaire indexen die we hebben aangebracht voor de testen te houden.

Daarnaast hebben we ervoor gekozen om geen tabellen samen te voegen of op te splitsen. Dit hebben we per query beargumenteerd waarom we van mening waren dat dit niet nodig was.