# Database Basics MS SQL Exam – 24 Jun 2018

Exam problems for the [“Database Basics” course @ SoftUni](https://softuni.bg/courses/databases-basics-ms-sql-server).

Submit your solutions in the SoftUni Judge system at <https://judge.softuni.bg/>

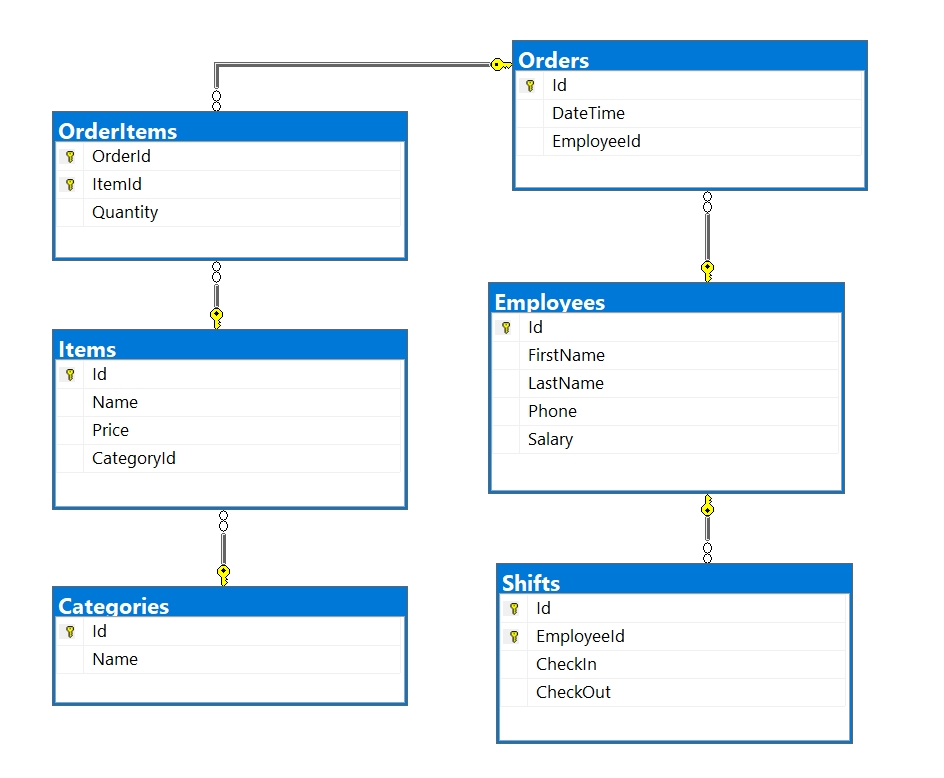
# Supermarket

After feeling extremely disappointed with your previous job at “**Krivodol Trip Service LLC**”, you have now started working for a new and much better company– “**Pustinqk Software**”. From the very beginning your new boss saw a huge potential in you and has assigned you a very exciting project. In **6 hours**, you must develop a complicated system for a small shop, which has now grown bigger.

# Your database must contain information about the employees and their work hours. You must also include information about the products and their orders.

# Section 1. DDL (30 pts)

You are given an E/R Diagram of the Trip Service:



Crеate a database called **Supermarket**. You need to create **6 tables**:

* **Categories** – contains information about the **item** **categories**.
* Items – contains information about the items and their categories.
* Orders – contains information about all of the store orders.
* OrderItems – contains information about every order’s items.
* Employees – contains information about the employees.
* Shifts – contains information about **check-in** tracking for **employees**.

### Categories

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Constraints** |
| Id | **Integer** from **0** to **2,147,483,647** | Unique table **identificator**, **Identity** |
| Name | **String** up to 30 symbols, Unicode | **NULL** is **not** allowed |

### Items

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Constraints** |
| Id | **Integer** from **0** to **2,147,483,647** | Unique table **identificator**, **Identity** |
| Name | **String** up to 30 symbols, Unicode | **NULL** is **not** allowed |
| Price | **Decimal** number with **two-digit** precision | **NULL** is **not** allowed |
| CategoryId | **Integer** from **0** to **2,147,483,647** | **NULL** is **not** allowed, Relationship with table Categories |

### Employees

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Constraints** |
| Id | **Integer** from **0** to **2,147,483,647** | Unique table **identificator**, **Identity** |
| FirstName | **String** up to 50 symbols, Unicode | **NULL** is **not** allowed |
| LastName | **String** up to 50 symbols, Unicode | **NULL** is **not** allowed |
| Phone | **String** with **exactly 12** symbols | **NULL** is **not** allowed |
| Salary | **Decimal** number with **two-digit** precision | **NULL** is **not** allowed |

### Orders

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Constraints** |
| Id | **Integer** from **0** to **2,147,483,647** | Unique table **identificator**, **Identity** |
| DateTime | **DateTime** | **NULL** is **not** allowed |
| EmployeeId | **Integer** from **0** to **2,147,483,647** | **NULL** is **not** allowed, Relationship with table Employees |

### OrderItems

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Constraints** |
| OrderId | **Integer** from **0** to **2,147,483,647** | **NULL** is **not** allowed, Relationship with table **Orders** |
| ItemId | **Integer** from **0** to **2,147,483,647** | **NULL** is **not** allowed, Relationship with table **Items** |
| Quantity | **Integer** from **0** to **2,147,483,647** | **NULL** is **not** allowed, must be **at least 1** |

### Shifts

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Constraints** |
| Id | **Integer** from **0** to **2,147,483,647** | Unique table **identificator**, **Identity** |
| EmployeeId | **Integer** from **0** to **2,147,483,647** | Unique table **identificator** , Relationship with table **Employees** |
| CheckIn | **DateTime** | **NULL** is **not** allowed |
| CheckOut | **DateTime** | **NULL** is **not** allowed, must be after CheckIn date |

## Database Design

Submit all of yours **create** **statements** to the **Judge** system.

**Solution:**

CREATE TABLE Categories

(

Id INT PRIMARY KEY IDENTITY,

[Name] NVARCHAR(30) NOT NULL

)

CREATE TABLE Items

(

Id INT PRIMARY KEY IDENTITY,

[Name] NVARCHAR(30) NOT NULL,

Price DECIMAL(18,2) NOT NULL,

CategoryId INT NOT NULL FOREIGN KEY REFERENCES Categories(Id)

)

CREATE TABLE Employees

(

Id INT PRIMARY KEY IDENTITY,

FirstName NVARCHAR(50) NOT NULL,

LastName NVARCHAR(50) NOT NULL,

Phone CHAR(12) NOT NULL,

Salary DECIMAL(18,2) NOT NULL,

)

CREATE TABLE Orders

(

Id INT PRIMARY KEY IDENTITY,

[DateTime] DATETIME NOT NULL,

EmployeeId INT NOT NULL FOREIGN KEY REFERENCES Employees(Id)

)

CREATE TABLE OrderItems

(

OrderId INT, ItemId INT,

CONSTRAINT PK\_OrderItems

PRIMARY KEY(OrderId, ItemId),

CONSTRAINT FK\_OrderItems\_Orders

FOREIGN KEY(OrderId)

REFERENCES Orders(Id),

CONSTRAINT FK\_OrderItems\_Items

FOREIGN KEY(ItemId)

REFERENCES Items(Id),

Quantity INT NOT NULL CHECK (Quantity > 0)

)

CREATE TABLE Shifts

(

Id INT IDENTITY,

EmployeeId INT ,

CONSTRAINT PK\_ShiftsIdEmployeeId

PRIMARY KEY(Id, EmployeeId),

CONSTRAINT FK\_ShiftsIdEmployeeId\_EmployeeId

FOREIGN KEY(EmployeeId)

REFERENCES Employees(Id),

CheckIn DATETIME NOT NULL,

CheckOut DATETIME NOT NULL,

CONSTRAINT CK\_TIMEDIFF CHECK(CheckOut>= CheckIn)

)

# Section 2. DML (10 pts)

**Before you start, you must import “**DataSet-Supermarket.sql**”. If you have created the structure correctly, the data should be successfully inserted without any errors.**

In this section, you have to do some data manipulations:

## Insert

**Insert** some sample data into the database. Write a query to add the following records into the corresponding tables. **All Ids should be auto-generated**.

**Solution:**

INSERT INTO Employees (FirstName, LastName, Phone, Salary)

VALUES

('Stoyan', 'Petrov', '888-785-8573', 500.25),

('Stamat', 'Nikolov', '789-613-1122', 999995.25),

('Evgeni', 'Petkov', '645-369-9517', 1234.51),

('Krasimir', 'Vidolov', '321-471-9982', 50.25)

INSERT INTO Items ([Name], Price, CategoryId)

VALUES

('Tesla battery', 154.25, 8),

('Chess', 30.25, 8),

('Juice', 5.32, 1),

('Glasses', 10, 8),

('Bottle of water', 1, 1)

**Employees**

|  |  |  |  |
| --- | --- | --- | --- |
| **FirstName** | **LastName** | **Phone** | **Salary** |
| Stoyan | Petrov | 888-785-8573 | 500.25 |
| Stamat | Nikolov | 789-613-1122 | 999995.25 |
| Evgeni | Petkov | 645-369-9517 | 1234.51 |
| Krasimir | Vidolov | 321-471-9982 | 50.25 |

**Items**

|  |  |  |
| --- | --- | --- |
| **Name** | **Price** | **CategoryId** |
| Tesla battery | 154.25 | 8 |
| Chess | 30.25 | 8 |
| Juice | 5.32 | 1 |
| Glasses | 10 | 8 |
| Bottle of water | 1 | 1 |

## Update

Make all items’ prices **27% more expensive** where the **category ID** is either **1**, **2** or **3.**

**Solution:**

UPDATE Items

SET Price\*=1.27

WHERE CategoryId BETWEEN 1 AND 3

## Delete

Delete all order items where the order id is 48 (be careful with the relationships)

**Solution:**

DELETE FROM OrderItems

WHERE OrderId = 48

# Section 3. Querying (40 pts)

**You need to start with a fresh dataset, so recreate your DB and import the sample data again (**DataSet-Supermarket.sql**).**

## Richest People

Select all **employees** who have a **salary** above **6500**. Order them by **first name**, then by **employee** **id**.

**Solution:**

SELECT Id, FirstName FROM Employees

WHERE Salary > 6500

ORDER BY FirstName, Id

### Example

|  |  |
| --- | --- |
| **Id** | **FirstName** |
| 19 | Arney |
| 32 | Arther |
| 2 | Celie |
| 11 | Emlynn |
| … | … |

## Cool Phone Numbers

Select all **full names** from employees,whose phone number start with ‘**3**’.

Order them by **first name** **(ascending)**, then by phone number **(ascending)**.

**Solution:**

SELECT CONCAT(FirstName, ' ', LastName) AS [Full Name],

Phone AS [Phone Number] FROM Employees

WHERE Phone LIKE '3%'

ORDER BY FirstName, Phone

### Example

|  |  |
| --- | --- |
| **Full Name** | **Phone Number** |
| Adolphe Leacock | 339-446-1263 |
| Audie Risebarer | 341-873-1275 |
| Demeter Langdale | 312-175-3209 |
| Jordanna Asmus | 323-785-5898 |
| … | … |

## Employee Statistics

Select all **employees** who have orders with the total count of the orders they processed. Order them by their **orders count (descending)**, then by **first name**. Select their **first name**, **last name** and **total count** of **orders**.

**Solution:**

SELECT e.FirstName, e.LastName, COUNT(e.Id) as [Count] FROM Employees AS e

JOIN Orders AS o ON e.Id = o.EmployeeId

GROUP BY e.FirstName, e.LastName

ORDER BY COUNT(e.Id) DESC, e.FirstName

### Example

|  |  |  |
| --- | --- | --- |
| **FirstName** | **LastName** | **Count** |
| Bart | Jozwiak | 123 |
| Beverlee | Raveau | 116 |
| Ashley | Topliss | 106 |
| Gayler | Wike | 103 |
| Celie | De Cruce | 96 |
| … | … | … |

## Hard Workers Club

Select all **employees** whose workday is **over 7 hours long on average**, based on their **check in/check out times**. Select their **first, last name** and **average** **work hours**.

Order them by **work hours** **(descending),** then by **employee ID**.

**Solution:**

SELECT e.FirstName,

e.LastName,

AVG(DATEDIFF(HOUR,CheckIn , CheckOut)) AS [Work Hours]

FROM Employees AS e

JOIN Shifts AS sh ON sh.EmployeeId = e.Id

GROUP BY e.FirstName, e.LastName, e.Id

HAVING AVG(DATEDIFF(HOUR,CheckIn , CheckOut)) > 7

ORDER BY AVG(DATEDIFF(HOUR,CheckIn , CheckOut)) DESC, e.Id

### Example

|  |  |  |
| --- | --- | --- |
| **FirstName** | **LastName** | **Work hours** |
| Gill | Wasiela | 9 |
| Celie | De Cruce | 8 |
| Jordanna | Asmus | 8 |
| Lucie | Dickinson | 8 |
| … | … | ... |

## The Most Expensive Order

Find the most expensive order. Select its **id** and total item price. Consider the item **quantity** when calculating the price.

**Solution:**

SELECT TOP(1) o.Id AS OrderId,

SUM((oi.Quantity \* i.Price)) AS TotalPrice

FROM Orders AS o

JOIN OrderItems AS oi ON o.Id = oi.OrderId

JOIN Items AS i ON i.Id = oi.ItemId

GROUP BY o.Id

ORDER BY SUM((oi.Quantity \* i.Price)) DESC

### Example

|  |  |
| --- | --- |
| **OrderId** | **TotalPrice** |
| 479 | 14087.84 |

## Rich Item, Poor Item

Find the top 10 **most expensive** and **cheapest** item in **each order.**

Order the results by **most expensive item’s price** (**descending),** then by order id (**ascending**)**.**

**Solution:**

SELECT TOP(10) o.Id AS OrderId,

MAX(i.Price) AS [ExpensivePrice],

MIN(i.Price) AS [CheapPrice]

FROM Orders AS o

JOIN OrderItems AS oi ON o.Id = oi.OrderId

JOIN Items AS i ON i.Id = oi.ItemId

GROUP BY o.Id

ORDER BY MAX(i.Price) DESC, o.Id

### Example

|  |  |  |
| --- | --- | --- |
| **OrderId** | **ExpensivePrice** | **CheapPrice** |
| 1 | 360.00 | 3.14 |
| 6 | 360.00 | 1.50 |
| 10 | 360.00 | 1.23 |
| 39 | 360.00 | 2.00 |
| … | … |  |

## Cashiers

Find all employees who have orders. Select their id, first name and last name. Order them by **employee id**.

**Solution:**

SELECT DISTINCT e.Id, e.FirstName, e.LastName FROM Employees AS e

JOIN Orders AS o ON e.Id = o.EmployeeId

ORDER BY e.Id

### Example

|  |  |  |
| --- | --- | --- |
| **Id** | **First Name** | **Last Name** |
| 2 | Celie | De Cruce |
| 5 | Lucie | Dickinson |
| 8 | Adaline | Gilogly |
| … | … | … |

## Lazy Employees

Find all employees, who have below 4 work hours per day.

Order them by employee id.

**Solution:**

SELECT DISTINCT e.Id, CONCAT(e.FirstName, ' ', e.LastName) AS [Full Name] FROM Employees AS e

JOIN Shifts AS sh ON sh.EmployeeId = e.Id

WHERE DATEDIFF(HOUR, sh.CheckIn, sh.CheckOut) < 4

ORDER BY e.Id

### Example

|  |  |
| --- | --- |
| **Id** | **Full Name** |
| 1 | Krishnah Lalor |
| 4 | Jasmine Forsdike |
| 7 | Ole De la Feld |
| … | … |

## Sellers

Find the top 10 employees with their full name, orders’ total price and item count.

Count only orders which were **ordered** **before 2018-06-15**.

Order them by **total sum (descending)**, then by **item count (descending)**

**Solution:**

SELECT CONCAT(e.FirstName, ' ', e.LastName) AS [Full Name],

SUM(oi.Quantity \* i.Price) AS [Total Price] ,

SUM(oi.Quantity) AS Items

FROM Employees AS e

JOIN Orders AS o ON e.Id = o.EmployeeId

JOIN OrderItems AS oi ON o.Id = oi.OrderId

JOIN Items AS i ON oi.ItemId = i.Id

WHERE o.[DateTime] < '2018-06-15'

GROUP BY CONCAT(e.FirstName, ' ', e.LastName)

ORDER BY SUM(oi.Quantity \* i.Price) DESC, SUM(oi.Quantity) DESC

### Example

|  |  |  |
| --- | --- | --- |
| **Full Name** | **Total Price** | **Items** |
| Bart Jozwiak | 37612.33 | 2497 |
| Adaline Gilogly | 26989.77 | 1765 |
| Celie De Cruce | 25692.80 | 1773 |
| Gayler Wike | 24754.87 | 2350 |
| Lucie Dickinson | 23707.26 | 1223 |
| … | … | … |

## Tough days

Find all records of the employees who don’t have orders and who work over 12 hours.

Select only their full name and day of the week.

Order the results by **employee id**.

**Note: By the American Standards, Sunday is the first day of week.**

**Solution:**

SELECT CONCAT(e.FirstName, ' ', e.LastName) as [Full Name],

DATENAME(weekday, s.CheckIn) AS [Day of week]

FROM Employees AS e

LEFT JOIN Orders AS o ON e.Id = o.EmployeeId

JOIN Shifts AS s ON s.EmployeeId = e.Id

WHERE o.Id IS NULL AND DATEDIFF(HOUR, s.CheckIn, s.CheckOut) > 12

ORDER BY e.Id

### Example

|  |  |
| --- | --- |
| **Full Name** | **Day of week** |
| Krishnah Lalor | Sunday |
| Jordanna Asmus | Monday |
| Ole De la Feld | Friday |
| Ole De la Feld | Thursday |
| … | … |

## Top Order per Employee

Find all information of the employees who have orders. Select their full name, duration of the work day (**in hours**) and total price of all sold products. Find only the **top orders** (top orders with highest total price).

Sort them by **full name** (ascending), **work hours** (descending) and **total price** (descending)

### Example

|  |  |  |
| --- | --- | --- |
| **Full Name** | **WorkHours** | **TotalPrice** |
| Adaline Gilogly | 5 | 9460.00 |
| Adolphe Leacock | 5 | 14087.84 |
| Anatola Lydon | 8 | 4090.80 |
| … | … | … |

## Average Profit per Day

Find the **average** **profit** for each day. Select the **day of month** and **average daily profit** of sold products.

Sort them by **day of month** (ascending) and format the profit to the **second digit** after the decimal point.

**Solution:**

SELECT DATEPART(DAY, o.DateTime) AS [Day],

FORMAT(AVG(oi.Quantity \* i.Price), 'N2') AS [Total profit]

FROM Orders AS o

JOIN OrderItems AS oi ON oi.OrderId = o.Id

JOIN Items AS i ON i.Id = oi.ItemId

GROUP BY DATEPART(DAY, o.DateTime)

ORDER BY DATEPART(DAY, o.DateTime)

### Example

|  |  |
| --- | --- |
| **Day** | **Total profit** |
| 1 | 254.79 |
| 3 | 211.49 |
| 4 | 115.89 |
| 5 | 83.26 |
| 6 | 111.47 |
| 7 | 101.49 |
| 8 | 140.65 |
| 10 | 90.17 |
| 11 | 281.59 |
| 12 | 162.31 |
| 13 | 127.65 |
| … | … |

## Top Products

Find information about **all products**. Select their name, category, how many of them were sold and the total profit they produced.

Sort them by **total profit** (**descending**) and **their count** (**descending**)

**Solution:**

SELECT i.[Name] AS [Item],

c.[Name] AS [Category],

SUM(oi.Quantity) AS [Count],

SUM(oi.Quantity) \* i.Price AS TotalPrice

FROM Items AS i

LEFT JOIN OrderItems AS oi ON oi.ItemId = i.Id

JOIN Categories AS c ON i.CategoryId = c.Id

GROUP BY i.[Name], c.[Name], i.Price

ORDER BY SUM(oi.Quantity) \* i.Price DESC, SUM(oi.Quantity) DESC

### Example

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Category** | **Count** | **TotalPrice** |
| TV | Miscellaneous | 308 | 110880.00 |
| Tires | Miscellaneous | 524 | 78600.00 |
| Mattress | Miscellaneous | 298 | 29800.00 |
| Camera | Miscellaneous | 352 | 28160.00 |
| … | … | … | … |

# Section 4. Programmability (20 pts)

## Promotion days

Create a **user defined function**, named **udf\_GetPromotedProducts(@CurrentDate, @StartDate, @EndDate, @Discount, @FirstItemId, @SecondItemId, @ThirdItemId)**, that receives a **current date**, a **start date** for the promotion, an **end date** for the promotion, a **discount**, a **first item id**, a **second item id** and **third item id**.

The function should print the discounted price of the items, based on these conditions:

* The first, second and third items must exist in the database.
* The current date must be between the start date and end date.

If both conditions are true, you must discount the price and print the following message in the format:

* “**{FirstItemName} price: {@FirstItemPrice} <-> {SecondItemName} price: {@SecondItemPrice} <-> {ThirdItemName} price: {@ThirdItemPrice}**”

If one of the items is not in the database, the function should return “One of the items does not exists!”  
If the current date is not between the start date and end date, the function should return “The current date is not within the promotion dates!”

**Note: Do not update any records in the database!**

**Solution:**

CREATE FUNCTION udf\_GetPromotedProducts

(

@CurrentDate DATE,

@StartDate DATE, @EndDate DATE,

@Discount INT, -- OR DECIMAL???

@FirstItemId INT,

@SecondItemId INT ,

@ThirdItemId INT

)

RETURNS VARCHAR(MAX)

AS

BEGIN

DECLARE @firstItem VARCHAR(MAX)

DECLARE @secondItem VARCHAR(MAX)

DECLARE @thirdItem VARCHAR(MAX)

DECLARE @firstItemPrice DECIMAL (18,2)

DECLARE @secondItemPrice DECIMAL (18,2)

DECLARE @thirdItemPrice DECIMAL (18,2)

DECLARE @result VARCHAR(MAX)

IF(NOT EXISTS(SELECT \* FROM Items

WHERE Id = @FirstItemId))

BEGIN

SET @result = 'One of the items does not exists!'

END

ELSE IF(NOT EXISTS(SELECT \* FROM Items

WHERE Id = @SecondItemId))

BEGIN

SET @result = 'One of the items does not exists!'

END

ELSE IF (NOT EXISTS(SELECT \* FROM Items

WHERE Id = @ThirdItemId))

BEGIN

SET @result = 'One of the items does not exists!'

END

ELSE IF(@CurrentDate < @StartDate OR @CurrentDate > @EndDate)

BEGIN

SET @result = 'The current date is not within the promotion dates!'

END

ELSE

BEGIN

SET @firstItem = (SELECT [Name] FROM Items

WHERE Id = @FirstItemId)

SET @secondItem = (SELECT [Name] FROM Items

WHERE Id = @SecondItemId)

SET @thirdItem = (SELECT [Name] FROM Items

WHERE Id = @ThirdItemId)

SET @firstItemPrice = ((SELECT Price FROM Items

WHERE Id IN (@FirstItemId)) - (SELECT Price FROM Items

WHERE Id IN (@FirstItemId)) \* @Discount / 100)

SET @secondItemPrice = ((SELECT Price FROM Items

WHERE Id IN (@SecondItemId)) - (SELECT Price FROM Items

WHERE Id IN (@SecondItemId)) \* @Discount / 100)

SET @thirdItemPrice = ((SELECT Price FROM Items

WHERE Id IN (@ThirdItemId)) - (SELECT Price FROM Items

WHERE Id IN (@ThirdItemId)) \* @Discount / 100)

SET @result = CONCAT(@firstItem, ' price: ',@firstItemPrice,' <-> ',@secondItem, ' price: ',@secondItemPrice, ' <-> ', @thirdItem, ' price: ',@thirdItemPrice)

END

RETURN @result

END

### Example:

|  |
| --- |
| **Query** |
| **SELECT dbo.udf\_GetPromotedProducts('2018-08-02', '2018-08-01', '2018-08-03',13, 3,4,5)** |
| **Output** |
| **Water price: 0.74 <-> Juice price: 1.31 <-> Ayran price: 4.35** |

|  |
| --- |
| **Query** |
| **SELECT dbo.udf\_GetPromotedProducts('2018-08-01', '2018-08-02', '2018-08-03',13,3 ,4,5)** |
| **Output** |
| The current date is not within the promotion dates! |

## Cancel order

Create a **user defined stored procedure**, named **usp\_CancelOrder(@OrderId, @CancelDate)**, that receives an **order id** and **date**, and attempts to **delete the current order**. An order will only be deleted if all of these conditions **pass**:

* If the **order** doesn’t exists, then it **cannot be deleted.** **Raise an error** with the message “The order does not exist!”
* If the **cancel date** is 3 days after the issue date, **raise an error** with the message “You cannot cancel the order!”

If all the above conditions pass, **delete the order**.

**Solution:**

CREATE PROCEDURE usp\_CancelOrder(@OrderId INT , @CancelDate DATE)

AS

BEGIN

IF(NOT EXISTS (SELECT \* FROM Orders

WHERE Id = @OrderId))

BEGIN

SELECT 'The order does not exist!'

END

ELSE IF(EXISTS(SELECT \* FROM Orders

WHERE DATEDIFF(DAY, [DateTime], @CancelDate) = 3))

BEGIN

SELECT 'You cannot cancel the order!'

END

ELSE

BEGIN

DELETE FROM OrderItems

WHERE OrderId = @OrderId

DELETE FROM Orders

WHERE Id = @OrderId

END

END

### Example usage:

|  |  |
| --- | --- |
| **Query** | **Output** |
| **EXEC usp\_CancelOrder 1, '2018-06-02'**  **SELECT COUNT(\*) FROM Orders**  **SELECT COUNT(\*) FROM OrderItems** | **998**  **2455** |
| **EXEC usp\_CancelOrder 1, '2018-06-15'** | **You cannot cancel the order!** |
| **EXEC usp\_CancelOrder 124231, '2018-06-15'** | **The order does not exist!** |