CREATE DATABASE Airport

USE Airport

--CREATE

CREATE TABLE Planes(

Id INT PRIMARY KEY IDENTITY,

[Name] VARCHAR(30) NOT NULL,

Seats INT NOT NULL,

[Range] INT NOT NULL

)

CREATE TABLE Flights(

Id INT PRIMARY KEY IDENTITY,

DepartureTime DATETIME2,

ArrivalTime DATETIME2,

Origin VARCHAR(50) NOT NULL,

Destination VARCHAR(50) NOT NULL,

PlaneId INT FOREIGN KEY REFERENCES Planes(Id) NOT NULL

)

CREATE TABLE Passengers(

Id INT PRIMARY KEY IDENTITY,

FirstName VARCHAR(30) NOT NULL,

LastName VARCHAR(30) NOT NULL,

Age INT NOT NULL,

[Address] VARCHAR(30) NOT NULL,

PassportId CHAR(11) NOT NULL

)

CREATE TABLE LuggageTypes(

Id INT PRIMARY KEY IDENTITY,

[Type] VARCHAR(30) NOT NULL

)

CREATE TABLE Luggages(

Id INT PRIMARY KEY IDENTITY,

LuggageTypeId INT FOREIGN KEY REFERENCES LuggageTypes(Id) NOT NULL,

PassengerId INT FOREIGN KEY REFERENCES Passengers(Id) NOT NULL

)

CREATE TABLE Tickets(

Id INT PRIMARY KEY IDENTITY,

PassengerId INT FOREIGN KEY REFERENCES Passengers(Id) NOT NULL,

FlightId INT FOREIGN KEY REFERENCES Flights(Id) NOT NULL,

LuggageId INT FOREIGN KEY REFERENCES Luggages(Id) NOT NULL,

Price DECIMAL(15,2) NOT NULL

)

--CREATE TABLE with 2 PK

CREATE TABLE LuggagePassengers(

LuggageId INT FOREIGN KEY REFERENCES Luggages(Id),

PassengerId INT FOREIGN KEY REFERENCES Passengers(Id),

CONSTRAINT PK\_CompositeLuggageIdPassengerId

PRIMARY KEY (LuggageId, PassengerId)

)

--CREATE NEW DATABASE

CREATE TABLE Students (

Id INT PRIMARY KEY IDENTITY,

FirstName NVARCHAR(30) NOT NULL,

MiddleName NVARCHAR(25),

LastName NVARCHAR(30) NOT NULL,

Age SMALLINT CHECK(Age BETWEEN 5 AND 100),

[Address] NVARCHAR(50),

Phone NCHAR(10)

)

CREATE TABLE Subjects (

Id INT PRIMARY KEY IDENTITY,

[Name] NVARCHAR(20) NOT NULL,

Lessons INT CHECK(Lessons > 0) NOT NULL

)

CREATE TABLE StudentsSubjects (

Id INT PRIMARY KEY IDENTITY,

StudentId INT FOREIGN KEY REFERENCES Students(Id) NOT NULL,

SubjectId INT FOREIGN KEY REFERENCES Subjects(Id) NOT NULL,

Grade DECIMAL(3,2) CHECK(Grade BETWEEN 2 AND 6) NOT NULL

)

CREATE TABLE Exams (

Id INT PRIMARY KEY IDENTITY,

[Date] DATETIME,

SubjectId INT FOREIGN KEY REFERENCES Subjects(Id) NOT NULL

)

CREATE TABLE StudentsExams (

StudentId INT FOREIGN KEY REFERENCES Students(Id) NOT NULL,

ExamId INT FOREIGN KEY REFERENCES Exams(Id) NOT NULL,

Grade DECIMAL(3,2) CHECK(Grade BETWEEN 2 AND 6) NOT NULL,

CONSTRAINT PK\_CompositeStudentIdExamId

PRIMARY KEY (StudentId, ExamId)

)

CREATE TABLE Jobs(

JobId INT PRIMARY KEY IDENTITY NOT NULL,

ModelId INT FOREIGN KEY REFERENCES Models(ModelId) NOT NULL,

[Status] NVARCHAR(11) NOT NULL DEFAULT 'Pending',

CHECK ([Status] IN ('Pending', 'In Progress', 'Finished')),

ClientId INT FOREIGN KEY REFERENCES Clients(ClientId) NOT NULL,

MechanicId INT FOREIGN KEY REFERENCES Mechanics(MechanicId),

IssueDate DATETIME2 NOT NULL,

FinishDate DATETIME2

)

CREATE TABLE Teachers (

Id INT PRIMARY KEY IDENTITY,

FirstName NVARCHAR(20) NOT NULL,

LastName NVARCHAR(20) NOT NULL,

[Address] NVARCHAR(20) NOT NULL,

Phone CHAR(10),

SubjectId INT FOREIGN KEY REFERENCES Subjects(Id) NOT NULL

)

CREATE TABLE StudentsTeachers (

StudentId INT FOREIGN KEY REFERENCES Students(Id) NOT NULL,

TeacherId INT FOREIGN KEY REFERENCES Teachers(Id) NOT NULL,

CONSTRAINT PK\_CompositeStudentIdTeacherId

PRIMARY KEY (StudentId, TeacherId)

)

--INSERT

INSERT INTO Planes([Name], Seats, [Range]) VALUES

('Airbus 336', 112, 5132),

('Airbus 330', 432, 5325),

('Boeing 369', 231, 2355),

('Stelt 297', 254, 2143),

('Boeing 338', 165, 5111),

('Airbus 558', 387, 1342),

('Boeing 128', 345, 5541)

--UPDATE

UPDATE Tickets

SET Price \*= 1.13

WHERE FlightId IN (SELECT Id FROM Flights WHERE Destination = 'Carlsbad')

UPDATE Jobs

SET [Status] = 'In Progress', MechanicId = 3

WHERE [Status] = 'Pending'

--DELETE

DELETE FROM Tickets

WHERE FlightId IN (SELECT Id FROM Flights WHERE Destination = 'Ayn Halagim')

DELETE FROM Flights

WHERE Destination = 'Ayn Halagim'

--DELETE

DELETE FROM StudentsTeachers

WHERE TeacherId IN (SELECT Id FROM Teachers WHERE Phone LIKE '%72%')

DELETE FROM Teachers

WHERE Id IN (SELECT Id FROM Teachers WHERE Phone LIKE '%72%')

--SELECT BY

SELECT \*

FROM Planes

WHERE [Name] LIKE '%tr%'

ORDER BY Id, [Name], Seats, [Range]

--SAME SELECT

SELECT \*

FROM Planes

WHERE CHARINDEX('tr', [Name]) > 0

ORDER BY Id, [Name], Seats, [Range]

SELECT t.FlightId, SUM(t.Price) AS Price

FROM Flights AS f

JOIN Tickets AS t ON t.FlightId = f.Id

GROUP BY t.FlightId

ORDER BY SUM(t.Price) DESC, t.FlightId

SELECT CONCAT(p.FirstName, ' ', p.LastName) AS [Full Name], f.Origin, f.Destination

FROM Passengers AS p

JOIN Tickets AS t ON t.PassengerId = p.Id

JOIN Flights AS f ON f.Id = t.FlightId

ORDER BY [Full Name], Origin, Destination

SELECT p.FirstName, p.LastName, p.Age

FROM Passengers AS p

LEFT JOIN Tickets AS t ON t.PassengerId = p.Id

WHERE t.Id IS NULL

ORDER BY p.Age DESC, p.FirstName, p.LastName

SELECT CONCAT(m.FirstName, ' ', m.LastName) AS [Mechanic], AVG(DAY(j.FinishDate) - DAY(j.IssueDate)) AS [Average Days]

FROM Mechanics AS m

JOIN Jobs AS j ON j.MechanicId = m.MechanicId

WHERE j.Status = 'Finished'

GROUP BY m.FirstName, m.LastName, m.MechanicId

ORDER BY m.MechanicId

SELECT **OpenDate** must be in format - **'dd-MM-yyyy'**

SELECT [Description], FORMAT(r.OpenDate, 'dd-MM-yyyy') AS OpenDate

FROM Reports AS r

WHERE EmployeeId IS NULL

ORDER BY r.OpenDate, [Description]

--SAME SELECT with CONVERT

SELECT [Description], CONVERT(varchar, r.OpenDate, 105) AS OpenDate

FROM Reports AS r

WHERE EmployeeId IS NULL

ORDER BY r.OpenDate, [Description]

SELECT **on their birthday**

SELECT u.Username, c.Name AS CategoryName

FROM Users AS u

JOIN Reports AS r ON r.UserId = u.Id

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

WHERE MONTH(r.OpenDate) = MONTH(u.Birthdate) AND DAY(r.OpenDate) = DAY(u.Birthdate)

ORDER BY u.Username, c.Name

--SAME with DATEPART

SELECT u.Username, c.Name AS CategoryName

FROM Users AS u

LEFT JOIN Reports AS r ON r.UserId = u.Id

LEFT JOIN Categories AS c ON c.Id = r.CategoryId

WHERE DATEPART(MONTH,u.Birthdate) = DATEPART(MONTH,r.OpenDate) AND DAY(u.Birthdate) = DAY(r.OpenDate)

ORDER BY u.Username, c.Name

**DISTINCT Id -Уникален**

Select the **top 5** repositories in terms of **count** of **commits**

SELECT TOP 5 r.Id, r.Name, COUNT(\*) AS Commits

FROM RepositoriesContributors AS rc

JOIN Repositories AS r ON r.Id = rc.RepositoryId

JOIN Commits AS c ON c.RepositoryId = r.Id

GROUP BY r.Id, r.Name

ORDER BY Commits DESC, r.Id, r.Name

-SELECT JOIN Select all of the **files**, which are NOT a **parent** to any other file. **JOIN SAME FILES 2 TIME**

SELECT f2.Id, f2.[Name], CONCAT(f2.Size, 'KB') AS Size

FROM Files AS f

RIGHT JOIN Files AS f2 ON f2.Id = f.ParentId

WHERE f.Id IS NULL

ORDER BY f2.Id, f2.[Name], f2.Size DESC

--SELECT WITH CAST **2 symbols after decimal point**

SELECT TOP 10 s.FirstName, s.LastName, CAST(AVG(se.Grade) AS DECIMAL(3,2)) AS Grade

FROM Students AS s

JOIN StudentsExams AS se ON se.StudentId = s.Id

GROUP BY s.FirstName, s.LastName

ORDER BY Grade DESC, s.FirstName, s.LastName

SELECT

\* FROM(

SELECT

p.PartId,

p.Description,

pn.Quantity AS Required,

p.StockQty AS [In Stock],

ISNULL(op.Quantity, 0) AS Ordered

FROM Parts AS p

LEFT JOIN PartsNeeded AS pn ON pn.PartId = p.PartId

LEFT JOIN Jobs AS j ON j.JobId = pn.JobId

LEFT JOIN Orders AS o ON o.JobId = j.JobId

LEFT JOIN OrderParts AS op ON op.OrderId = o.OrderId

WHERE j.Status <> 'Finished' AND (o.Delivered IS NULL OR o.Delivered = 0)

) AS [PartsQuantitySubQuery]

WHERE [In Stock] + [Ordered] < [Required]

ORDER BY PartId

SELECT how many **unique** users each of them has served to

SELECT CONCAT(e.FirstName, ' ', e.LastName) AS FullName, COUNT(u.Id) AS UsersCount

FROM Employees AS e

LEFT JOIN Reports AS r ON r.EmployeeId = e.Id

LEFT JOIN Users AS u ON u.Id = r.UserId

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

ORDER BY UsersCount DESC, FullName

Select all **mechanics** without active **jobs** (**include mechanics which don’t have any job assigned or all of their jobs are finished**). Order by ID (ascending).

SELECT CONCAT(m.FirstName, ' ', m.LastName) AS [Available]

FROM Mechanics AS m

WHERE(m.MechanicId) NOT IN(SELECT MechanicId

FROM Jobs

WHERE Status = 'In Progress'

GROUP BY MechanicId)

ORDER BY m.MechanicId

SELECT CONCAT(FirstName, ' ', LastName) AS [Available] FROM Mechanics

WHERE(MechanicId) NOT IN(SELECT MechanicId FROM Jobs

WHERE Status = 'In Progress'

GROUP BY MechanicId)

ORDER BY MechanicId

--SELECT **NOTE**: If the middle name is null you have to concatenate the first name and last name **separated with single space**.

SELECT CONCAT(s.FirstName, ' ', IIF(s.MiddleName IS NULL, '', s.MiddleName + ' '), s.LastName) AS [Full Name]

FROM Students AS s

LEFT JOIN StudentsSubjects AS sub ON sub.StudentId = s.Id

WHERE sub.SubjectId IS NULL

ORDER BY [Full Name]

SELECT

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

pl.Name AS [Plane Name],

CONCAT(f.Origin, ' - ', f.Destination) AS Trip,

lt.Type AS [Luggage Type]

FROM Passengers AS p

JOIN Tickets AS t ON t.PassengerId = p.Id

JOIN Flights AS f ON f.Id = t.FlightId

JOIN Planes AS pl ON pl.Id = f.PlaneId

JOIN Luggages AS l ON l.Id = t.LuggageId

JOIN LuggageTypes AS lt ON lt.Id = l.LuggageTypeId

ORDER BY [Full Name], [Plane Name], Origin, Destination, [Luggage Type]

SELECT pl.Name, pl.Seats, COUNT(p.Id) AS [Passengers Count]

FROM Planes AS pl

LEFT JOIN Flights AS f ON f.PlaneId = pl.Id

LEFT JOIN Tickets AS t ON t.FlightId = f.Id

LEFT JOIN Passengers AS p ON p.Id = t.PassengerId

GROUP BY pl.Name, pl.Seats

ORDER BY [Passengers Count] DESC, pl.Name, pl.Seats

**SELECT HARDER**

SELECT

IIF(CONCAT(e.FirstName, ' ', e.LastName) = '', 'None', CONCAT(e.FirstName, ' ', e.LastName)) AS Employee,

IIF(d.Name IS NULL, 'None', d.Name) AS Department,

IIF(c.Name IS NULL, 'None', c.Name) AS Category,

IIF(r.Description IS NULL, 'None', r.Description) AS [Description],

IIF(r.OpenDate IS NULL, 'None', CONVERT(varchar, r.OpenDate, 104)) AS OpenDate,

IIF(s.Label IS NULL, 'None', s.Label) AS [Status],

IIF(u.[Name] IS NULL, 'None', u.[Name]) AS [User]

FROM Reports AS r

LEFT JOIN Employees AS e ON e.Id = r.EmployeeId

LEFT JOIN Departments AS d ON d.Id = e.DepartmentId

LEFT JOIN Categories AS c ON c.Id = r.CategoryId

LEFT JOIN [Status] AS s ON s.Id = r.StatusId

LEFT JOIN Users AS u ON u.Id = r.UserId

ORDER BY e.FirstName DESC, e.LastName DESC, Department, Category, [Description], OpenDate, [Status], [User]

--SAME ISNULL?

SELECT

ISNULL(e.FirstName + ' ' + e.LastName, 'None') AS Employee,

ISNULL(d.Name, 'None') AS Department,

ISNULL(c.Name, 'None') AS Category,

r.Description AS [Description],

FORMAT(r.OpenDate, 'dd.MM.yyy') AS OpenDate,

s.Label AS [Status],

ISNULL(u.Name, 'None') AS [User]

FROM Reports AS r

LEFT JOIN Employees AS e ON e.Id = r.EmployeeId

LEFT JOIN Departments AS d ON d.Id = e.DepartmentId

LEFT JOIN Categories AS c ON c.Id = r.CategoryId

LEFT JOIN [Status] AS s ON s.Id = r.StatusId

LEFT JOIN Users AS u ON u.Id = r.UserId

ORDER BY e.FirstName DESC, e.LastName DESC, Department, Category, [Description], r.OpenDate, [Status], [User]

Second Highest Grade

SELECT DISTINCT k.FirstName, k.LastName, k.Grade AS Grade

FROM (

SELECT s.FirstName, s.LastName, ss.Grade, DENSE\_RANK() OVER (PARTITION BY s.FirstName, s.LastName ORDER BY ss.Grade DESC, ss.Id) AS GradeRank

FROM Students AS s

RIGHT JOIN StudentsSubjects AS ss ON ss.StudentId = s.Id

) AS k

WHERE k.GradeRank = 2

ORDER BY k.FirstName, k.LastName

## 3rd Highest Salary

SELECT DISTINCT k.DepartmentID, k.Salary AS ThirdHighestSalary

FROM (

SELECT DepartmentID, Salary, DENSE\_RANK() OVER (PARTITION BY DepartmentID ORDER BY Salary DESC) AS SalaryRank

FROM Employees

) AS k

WHERE k.SalaryRank = 3

--CREATE FUNCTION

CREATE FUNCTION udf\_GetCost(@JobId INT)

RETURNS DECIMAL(8,2)

AS

BEGIN

DECLARE @TotalCost DECIMAL(8, 2)

SET @TotalCost = (

SELECT SUM(p.Price \* op.Quantity)

FROM Jobs AS j

JOIN Orders AS o ON o.JobId = j.JobId

JOIN OrderParts AS op ON op.OrderId = o.OrderId

JOIN Parts AS p ON p.PartId = op.PartId

WHERE j.JobId = @JobId)

IF (@TotalCost IS NULL)

BEGIN

RETURN 0;

END

RETURN @TotalCost

END

SELECT dbo.udf\_GetCost(3)

CREATE FUNCTION udf\_CalculateTickets(@origin VARCHAR(50), @destination VARCHAR(50), @peopleCount INT)

RETURNS VARCHAR(50)

AS

BEGIN

IF (@peopleCount <= 0)

BEGIN

RETURN 'Invalid people count!'

END

DECLARE @flightId INT = (SELECT TOP 1 Id FROM Flights WHERE Origin = @origin AND Destination = @destination)

IF (@flightId IS NULL)

BEGIN

RETURN 'Invalid flight!'

END

DECLARE @pricePerPerson DECIMAL(15,2) = (SELECT TOP 1 Price FROM Tickets WHERE FlightId = @flightId)

DECLARE @totalPrice DECIMAL(18,2) = @pricePerPerson \* @peopleCount

RETURN CONCAT('Total price ', @totalPrice)

END

SELECT dbo.udf\_CalculateTickets('Kolyshley','Rancabolang', 33)

SELECT dbo.udf\_CalculateTickets('Kolyshley','Rancabolang', -1)

SELECT dbo.udf\_CalculateTickets('Invalid','Rancabolang', 33)

--CREATE FUNCTION

CREATE FUNCTION udf\_HoursToComplete(@StartDate DATETIME, @EndDate DATETIME)

RETURNS INT

BEGIN

IF (@StartDate IS NULL)

BEGIN

RETURN 0

END

IF (@EndDate IS NULL)

BEGIN

RETURN 0

END

RETURN DATEDIFF(HOUR, @StartDate, @EndDate)

END

SELECT dbo.udf\_HoursToComplete(OpenDate, CloseDate) AS TotalHours

FROM Reports

-CREATE FUNCTION

CREATE FUNCTION udf\_ExamGradesToUpdate(@studentId INT, @grade DECIMAL(3, 2))

RETURNS VARCHAR(MAX)

AS

BEGIN

IF (@grade > 6.00)

BEGIN

RETURN 'Grade cannot be above 6.00!'

END

DECLARE @studentFirstName NVARCHAR(30) = (

SELECT TOP 1 s.FirstName

FROM Students AS s

JOIN StudentsExams AS se ON se.StudentId = s.Id

WHERE (Grade BETWEEN @grade AND (@grade + 0.50)) AND StudentId = @studentId

)

IF (@studentFirstName IS NULL)

BEGIN

RETURN 'The student with provided id does not exist in the school!'

END

DECLARE @countOfStudents INT = (

SELECT COUNT(s.FirstName)

FROM Students AS s

JOIN StudentsExams AS se ON se.StudentId = s.Id

WHERE (Grade BETWEEN @grade AND (@grade + 0.50)) AND StudentId = @studentId

GROUP BY s.FirstName

)

RETURN CONCAT('You have to update ', @countOfStudents, ' grades for the student ', @studentFirstName)

END

--OR

CREATE FUNCTION udf\_ExamGradesToUpdate(@studentId INT, @grade DECIMAL(3,2))

RETURNS NVARCHAR(MAX)

BEGIN

DECLARE @studentFirstName NVARCHAR(30) = (SELECT FirstName FROM Students WHERE Id = @studentId )

IF (@studentFirstName IS NULL)

BEGIN

RETURN 'The student with provided id does not exist in the school!'

END

IF (@grade > 6)

BEGIN

RETURN 'Grade cannot be above 6.00!'

END

DECLARE @count INT = (SELECT COUNT(\*)

FROM StudentsExams

WHERE StudentId = @studentId AND Grade BETWEEN @grade AND (@grade + 0.50))

RETURN CONCAT('You have to update ', @count,' grades for the student ', @studentFirstName)

END

SELECT dbo.udf\_ExamGradesToUpdate(12, 6.20)

SELECT dbo.udf\_ExamGradesToUpdate(12, 5.50)

SELECT dbo.udf\_ExamGradesToUpdate(121, 5.50)

-CREATE PROC

CREATE PROC usp\_CancelFlights

AS

BEGIN

UPDATE Flights

SET DepartureTime = NULL, ArrivalTime = NULL

WHERE DATEDIFF(SECOND, DepartureTime, ArrivalTime) > 0

END

EXEC dbo.usp\_CancelFlights

CREATE PROC usp\_ExcludeFromSchool @StudentId INT

AS

BEGIN

DECLARE @studentCount BIT = (SELECT COUNT(\*) FROM Students WHERE Id = @StudentId)

IF(@studentCount = 0)

BEGIN

RAISERROR ('This school has no student with the provided id!', 16, 1)

RETURN

END

DELETE FROM StudentsExams WHERE StudentId = @StudentId

DELETE FROM StudentsSubjects WHERE StudentId = @StudentId

DELETE FROM StudentsTeachers WHERE StudentId = @StudentId

DELETE FROM Students WHERE Id = @StudentId

END

EXEC usp\_ExcludeFromSchool 123453

SELECT COUNT(\*) FROM Students

--CREATE PROC Create a **stored procedure** with the name **usp\_AssignEmployeeToReport**(@E**mployeeId INT**, @**ReportI**d INT) that receives an **employee's Id** and a **report's Id** and assigns the employee to the report **only if** the department of the employee and the department of the report's category are the same. Otherwise throw an **exception** with message: "Employee doesn't belong to the appropriate department!".

CREATE PROC usp\_AssignEmployeeToReport @EmployeeId INT, @ReportId INT

AS

BEGIN

DECLARE @result VARCHAR(MAX) = (SELECT e.Id

FROM Employees AS e

JOIN Categories AS c ON c.DepartmentId = e.DepartmentId

JOIN Reports AS r ON r.CategoryId = c.Id

WHERE c.DepartmentId = e.DepartmentId AND e.Id = @EmployeeId AND r.Id = @ReportId)

IF(@result IS NULL)

BEGIN

RAISERROR ('Employee doesn''t belong to the appropriate department!', 16, 1)

RETURN

END

UPDATE Reports

SET EmployeeId = @EmployeeId

END

--SAME

CREATE PROC usp\_AssignEmployeeToReport @EmployeeId INT, @ReportId INT

AS

BEGIN

DECLARE @EmployeeDepartmentId INT = (

SELECT DepartmentId FROM Employees WHERE Id = @EmployeeId

)

DECLARE @ReportDepartmentId INT = (

SELECT c.DepartmentId

FROM Reports AS r

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

WHERE r.Id = @ReportId

)

IF(@EmployeeDepartmentId != @ReportDepartmentId)

BEGIN

RAISERROR('Employee doesn''t belong to the appropriate department!', 16, 1)

RETURN

END

UPDATE Reports

SET EmployeeId = @EmployeeId

WHERE Id = @ReportId

END

CREATE PROCEDURE usp\_PlaceOrder @jobId INT, @partSerialNumber VARCHAR(50), @quantity INT

AS

BEGIN

IF (@quantity <= 0)

BEGIN

THROW 50012, 'Part quantity must be more than zero!', 1

END

IF((SELECT [Status] FROM [Jobs]

WHERE [JobId] = @jobId) = 'Finished')

BEGIN

THROW 50011, 'This job is not active!', 1

END

DECLARE @jobIdDb INT = (SELECT [JobId] FROM [Jobs]

WHERE [JobId] = @jobId)

IF(@jobIdDb IS NULL)

BEGIN

THROW 50013, 'Job not found!', 1

END

DECLARE @partId INT = (SELECT [PartId] FROM [Parts]

WHERE [SerialNumber] = @partSerialNumber)

IF(@partId IS NULL)

BEGIN

THROW 50014, 'Part not found!', 1

END

IF ((SELECT [OrderId] FROM [Orders]

WHERE [JobId] = @jobId AND [IssueDate] IS NULL) IS NULL)

BEGIN

INSERT INTO [Orders]([JobId], [IssueDate], [Delivered])

VALUES

(@jobId, NULL, 0)

END

DECLARE @orderId INT = (SELECT [OrderId] FROM [Orders]

WHERE [JobId] = @jobId AND [IssueDate] IS NULL)

DECLARE @orderedPartQuantity INT = (SELECT [Quantity] FROM [OrderParts]

WHERE [OrderId] = @orderId AND [PartId] = @partId)

IF (@orderedPartQuantity IS NULL)

BEGIN

INSERT INTO [OrderParts]([OrderId], [PartId], [Quantity])

VALUES

(@orderId, @partId, @quantity)

END

ELSE

BEGIN

UPDATE [OrderParts]

SET [Quantity] += @quantity

WHERE [OrderId] = @orderId AND [PartId] = @partId

END

END

-------------------HARDEST PROBLEM

DECLARE @userGameId INT = (SELECT Id FROM UsersGames WHERE UserId = 9 AND GameId = 87)

DECLARE @stamatCash DECIMAL (15,2) = (SELECT Cash FROM UsersGames WHERE Id = @userGameId)

DECLARE @itemsPrice DECIMAL (15,2) = (SELECT SUM(Price) AS TotalPrice FROM Items WHERE MinLevel BETWEEN 11 AND 12)

IF (@stamatCash >= @itemsPrice)

BEGIN

BEGIN TRANSACTION

UPDATE UsersGames

Set Cash -= @itemsPrice

WHERE Id = @userGameId

INSERT INTO UserGameItems (ItemId, UserGameId)

SELECT Id, @userGameId FROM Items WHERE MinLevel BETWEEN 11 AND 12

COMMIT

END

SET @stamatCash = (SELECT Cash FROM UsersGames WHERE Id = @userGameId)

SET @itemsPrice = (SELECT SUM(Price) AS TotalPrice FROM Items WHERE MinLevel BETWEEN 19 AND 21)

IF (@stamatCash >= @itemsPrice)

BEGIN

BEGIN TRANSACTION

UPDATE UsersGames

Set Cash -= @itemsPrice

WHERE Id = @userGameId

INSERT INTO UserGameItems (ItemId, UserGameId)

SELECT Id, @userGameId FROM Items WHERE MinLevel BETWEEN 19 AND 21

COMMIT

END

SELECT i.Name AS [Item Name]

FROM Users AS u

JOIN UsersGames AS ug ON ug.UserId = u.Id

JOIN Games AS g ON g.Id = ug.GameId

JOIN UserGameItems AS ugi ON ugi.UserGameId= ug.Id

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

WHERE u.Username = 'Stamat' AND g.Name = 'Safflower'

ORDER BY i.Name