# Database Basics MSSQL Exam – 17 Feb 2019

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/>

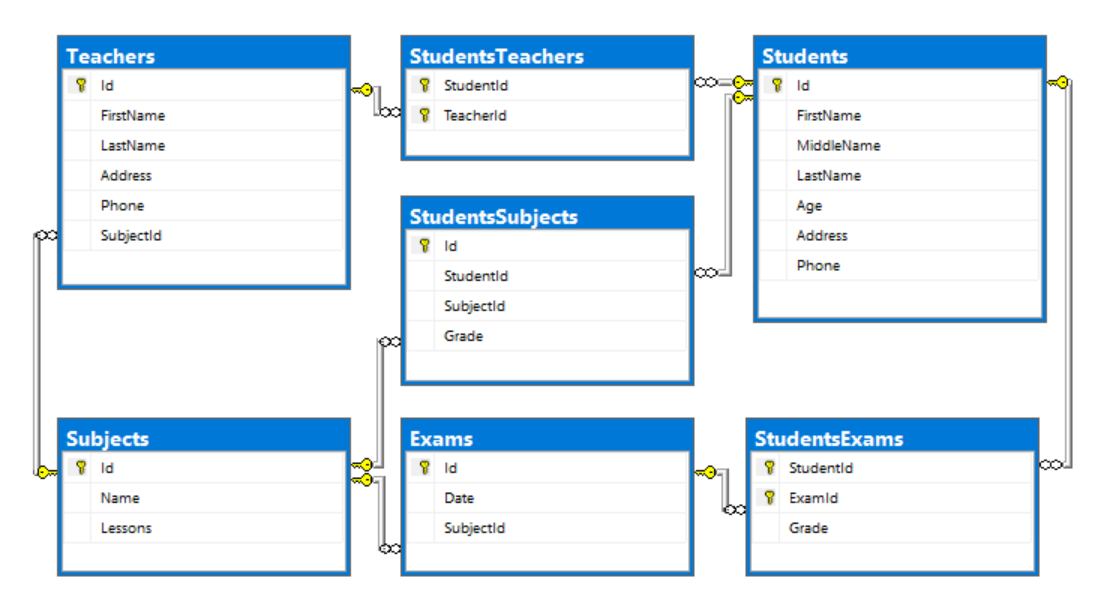
# School

Don’t be so stressed! Today you must build a very simple school system and execute some queries over it to check if it works correctly. From the very beginning **SoftUni** 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 school.

# Your database must contain information about the students with their teachers and exams. Also, it must contain information about the subjects in the school

# Section 1. DDL (30 pts)

You are given an E/R Diagram of the School:



Crеate a database called **School**. You need to create **7 tables**:

* **Students** – contains information about the students.
* Subjects – contains information about the subjects.
* StudentsSubjects – contains information about every student’s subjects.
* Exams – contains information about the exams.
* StudentsExams – contains information about every student’s exams.
* Teachers – contains information about the teachers.
* StudentsTeachers – contains information about every student’s teachers.

### Students

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Constraints** |
| Id | **Integer** from **0** to **2,147,483,647** | Unique table **identificator**, **Identity** |
| FirstName | **String** up to **30** symbols, Unicode | **NULL** is **not** allowed |
| MiddleName | **String** up to **25** symbols, Unicode | None |
| LastName | **String** up to **30** symbols, Unicode | **NULL** is **not** allowed |
| Age | **Integer** from **5** to **100** | **Negative or zero numbers** are **not allowed** |
| Address | **String** up to **50** symbols, Unicode | None |
| Phone | **String** with **exactly** 10 symbols, Unicode | None |

### Subjects

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Constraints** |
| Id | **Integer** from **0** to **2,147,483,647** | Unique table **identificator**, **Identity** |
| Name | **String** up to **20** symbols, Unicode | **NULL** is **not** allowed |
| Lessons | **Integer** must be more than 0 | **NULL** is **not** allowed |

### StudentsSubjects

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Constraints** |
| Id | **Integer** from **0** to **2,147,483,647** | Unique table **identificator**, **Identity** |
| StudentId | **Integer** from **0** to **2,147,483,647** | **NULL** is **not** allowed, Relationship with table **Students** |
| SubjectId | **Integer** from **0** to **2,147,483,647** | **NULL** is **not** allowed, Relationship with table **Subjects** |
| Grade | **Decimal** number with **two-digit** precision | **Grade** must be between 2 and 6, **NULL** is **not** allowed |

### Exams

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Constraints** |
| Id | **Integer** from **0** to **2,147,483,647** | Unique table **identificator**, **Identity** |
| Date | **DateTime** | None |
| SubjectId | **Integer** from **0** to **2,147,483,647** | **NULL** is **not** allowed, Relationship with table **Subjects** |

### StudentsExams

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Constraints** |
| StudentId | **Integer** from **0** to **2,147,483,647** | **NULL** is **not** allowed, Relationship with table **Students** |
| ExamId | **Integer** from **0** to **2,147,483,647** | **NULL** is **not** allowed, Relationship with table **Exams** |
| Grade | **Decimal** number with **two-digit** precision | **Grade** must be between 2 and 6, **NULL** is **not** allowed |

### Teachers

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Constraints** |
| Id | **Integer** from **0** to **2,147,483,647** | Unique table **identificator**, **Identity** |
| FirstName | **String** up to **20** symbols, Unicode | **NULL** is **not** allowed |
| LastName | **String** up to **20** symbols, Unicode | **NULL** is **not** allowed |
| Address | **String** up to **20** symbols, Unicode | **NULL** is **not** allowed |
| Phone | **String** with **exactly** **10** symbols | None |
| SubjectId | **Integer** from **0** to **2,147,483,647** | **NULL** is **not** allowed, Relationship with table **Subjects** |

### StudentsTeachers

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Constraints** |
| StudentId | **Integer** from **0** to **2,147,483,647** | **NULL** is **not** allowed, Relationship with table **Students** |
| TeacherId | **Integer** from **0** to **2,147,483,647** | **NULL** is **not** allowed, Relationship with table **Teachers** |

## Database Design

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

**Solution:**

CREATE TABLE Students

(

Id INT PRIMARY KEY IDENTITY,

FirstName NVARCHAR(30) NOT NULL,

MiddleName NVARCHAR(25) ,

LastName NVARCHAR(30) NOT NULL,

Age INT CHECK (Age >= 5 AND Age <= 100), -- including 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 >= 2 AND Grade <= 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,\_

ExamId INT,

CONSTRAINT PK\_StudentsExams

PRIMARY KEY(StudentId, ExamId),

CONSTRAINT FK\_StudentsExams\_Students

FOREIGN KEY(StudentId)

REFERENCES Students(Id),

CONSTRAINT FK\_StudentsExams\_Exams

FOREIGN KEY(ExamId)

REFERENCES Exams(Id),

Grade DECIMAL(3,2) CHECK (Grade >= 2 AND Grade <= 6 ) NOT NULL

)

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,\_

TeacherId INT,

CONSTRAINT PK\_StudentsTeachers

PRIMARY KEY(StudentId, TeacherId),

CONSTRAINT FK\_StudentsTeachers\_Students

FOREIGN KEY(StudentId)

REFERENCES Students(Id),

CONSTRAINT FK\_StudentsTeachers\_Teachers

FOREIGN KEY(TeacherId)

REFERENCES Teachers(Id)

)

# Section 2. DML (10 pts)

**Before you start, you must import “**DataSet-School.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 Teachers

(

FirstName,

LastName,

[Address],

Phone,

SubjectId

)

VALUES

('Ruthanne', 'Bamb', '84948 Mesta Junction', '3105500146', 6),

('Gerrard', 'Lowin', '370 Talisman Plaza', '3324874824', 2),

('Merrile', 'Lambdin', '81 Dahle Plaza', '4373065154', 5),

('Bert', 'Ivie', '2 Gateway Circle', '4409584510', 4)

INSERT INTO Subjects([Name], Lessons)

VALUES

('Geometry', 12),

('Health', 10),

('Drama', 7),

('Sports' ,9)

**Teachers**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **FirstName** | **LastName** | **Address** | **Phone** | **SubjectId** |
| Ruthanne | Bamb | 84948 Mesta Junction | 3105500146 | 6 |
| Gerrard | Lowin | 370 Talisman Plaza | 3324874824 | 2 |
| Merrile | Lambdin | 81 Dahle Plaza | 4373065154 | 5 |
| Bert | Ivie | 2 Gateway Circle | 4409584510 | 4 |

**Subjects**

|  |  |
| --- | --- |
| **Name** | **Lessons** |
| Geometry | 12 |
| Health | 10 |
| Drama | 7 |
| Sports | 9 |

## Update

Make all grades 6.00, where the subject id is 1 or 2, if the **grade** is above or equal to 5.50

**Solution:**

UPDATE StudentsSubjects

SET Grade = 6.00

WHERE SubjectId IN (1,2) AND Grade >= 5.50

## Delete

Delete all teachers, whose phone number contains ‘72’.

**Solution:**

DELETE FROM StudentsTeachers

WHERE TeacherId IN

(

SELECT Id FROM Teachers

WHERE Phone LIKE '%72%'

)

DELETE FROM Teachers

WHERE Phone LIKE '%72%'

# Section 3. Querying (40 pts)

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

## Teen Students

Select all **students** who are teenagers (their age is above or equal to 12). Order them by **first name (alphabetically)**, then by **last name (alphabetically)**. Select their first name, last name and their age.

**Solution:**

SELECT FirstName, LastName, Age FROM Students

WHERE Age>=12

ORDER BY FirstName, LastName

### Example

|  |  |  |
| --- | --- | --- |
| **FirstName** | **LastName** | **Age** |
| Agace | Sneddon | 12 |
| Andres | Colliard | 12 |
| Brose | Yeats | 13 |
| Casper | Tite | 12 |
| … | … | … |

## Cool Addresses

Select all **full names** from students,whose address text contains ‘road’.

Order them by **first name** **(alphabetically)**, then by **last name** **(alphabetically),** then by address text **(alphabetically)**.

**Solution:**

SELECT CONCAT(FirstName, ' ', MiddleName, ' ' ,LastName) AS [Full Name], [Address] FROM Students

WHERE Address LIKE '%road%'

ORDER BY FirstName, LastName, [Address]

### Example

|  |  |
| --- | --- |
| **Full Name** | **Address** |
| Clywd Jon Dyett | 1513 Lien Road |
| Garnet Lax De Cleyne | 91 Maple Road |
| Harland Trelevan Samber | 89863 Leroy Road |
| Lock Kenford Houlaghan | 3 Hovde Road |
| … | … |

## 42 Phones

Select students with middle names whose phones starts with **42**. Select their **first name**, **address** and **phone** **number**. Order them by first name **alphabetically**.

**Solution:**

SELECT FirstName, [Address], Phone FROM Students

WHERE MiddleName IS NOT NULL AND Phone LIKE '42%'

ORDER BY FirstName

### Example

|  |  |  |
| --- | --- | --- |
| **FirstName** | **Address** | **Phone** |
| Chloe | 520 Sauthoff Pass | 4216471468 |
| Freddie | 5 Basil Junction | 4205378077 |
| … | … | … |

## Students Teachers

Select all students and the count of teachers each one has.

**Solution:**

SELECT FirstName, LastName, COUNT(st.TeacherId) AS TeachersCount FROM Students AS s

JOIN StudentsTeachers AS st ON s.Id = st.StudentId

GROUP BY FirstName, LastName

### Example

|  |  |  |
| --- | --- | --- |
| **FirstName** | **LastName** | **TeachersCount** |
| Sandy | Abbison | 10 |
| Baxter | Abrahart | 13 |
| Demott | Addison | 13 |
| Deane | Adess | 10 |
| … | … | ... |

## Subjects with Students

Select all teachers’ full names and the subjects they teach with the count of lessons in each. Finally select the count of students each teacher has. Order them by students count descending, full name (ascending) and subjects (ascending).

**Solution:**

SELECT CONCAT(t.FirstName, ' ', t.LastName) AS [FullName],

CONCAT(s.[Name], '-', s.Lessons) AS [Subjects],

COUNT(st.StudentId) AS Students

FROM Teachers AS t

JOIN Subjects AS s ON t.SubjectId = s.Id

JOIN StudentsTeachers AS st ON st.TeacherId = t.Id

GROUP BY t.[FirstName], t.[LastName], s.[Name], s.[Lessons]

ORDER BY Students DESC, [FullName], [Subjects]

### Example

|  |  |  |
| --- | --- | --- |
| **FullName** | **Subjects** | **Students** |
| Rona Wollard | Physics-12 | 90 |
| Salvador Depport | French-15 | 90 |
| Ruthanne Bamb | Biology-12 | 90 |
| Merrile Lambdin | English-7 | 90 |
| Ezechiel Dalinder | Poetry-10 | 80 |
| … | … | … |

## Students to Go

Find all students, who have not attended an exam. Select their full name (first name + last name).

Order the results by full name (**ascending**)**.**

**Solution:**

SELECT CONCAT(s.FirstName, ' ', s.LastName) AS [Full Name] FROM StudentsExams AS se

RIGHT JOIN Students AS s ON s.Id = se.StudentId

WHERE se.StudentId IS NULL

ORDER BY [Full Name]

### Example

|  |
| --- |
| **Full Name** |
| Bernardine Purrier |
| … |

## Busiest Teachers

Find top 10 teachers with most students they teach. Select their first name, last name and the amount of students they have. Order them by students count (**descending**), then by first name (**ascending**), then by last name (**ascending**).

**Solution:**

SELECT TOP(10) t.FirstName, t.LastName, COUNT(st.StudentId) AS StudentsCount

FROM Teachers AS t

JOIN Subjects AS s ON t.SubjectId = s.Id

JOIN StudentsTeachers AS st ON st.TeacherId = t.Id

GROUP BY t.[FirstName], t.[LastName], s.[Name], s.[Lessons]

ORDER BY StudentsCount DESC, t.FirstName, t.LastName

### Example

|  |  |  |
| --- | --- | --- |
| **FirstName** | **LastName** | **StudentsCount** |
| Merrile | Lambdin | 90 |
| Rona | Wollard | 90 |
| Ruthanne | Bamb | 90 |
| … | … |  |

## Top Students

Find top 10 students, who have highest average grades from the exams.

Format the grade, two symbols after the decimal point.

Order them by grade (**descending**), then by first name (**ascending**), then by last name (**ascending**)

**Solution:**

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

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

GROUP BY s.FirstName, s.LastName

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

### Example

|  |  |  |
| --- | --- | --- |
| **First Name** | **Last Name** | **Grade** |
| Lurlene | Orgee | 6.00 |
| Ivy | Bilovsky | 5.70 |
| Chariot | Giacobbo | 5.50 |
| … | … |  |

## Second Highest Grade

Find the second highest grade per student from all subjects. Sort them by first name (**ascending**), then by last name (**ascending**).

**Solution:**

SELECT r.FirstName, r.LastName, r.Grade FROM (SELECT s.FirstName,

s.LastName,

se.Grade ,

ROW\_NUMBER()OVER(PARTITION BY s.FirstName ORDER BY Grade DESC) AS Ranking

FROM Students AS s

JOIN StudentsSubjects AS se ON se.StudentId = s.Id ) AS r

WHERE r.Ranking = 2

ORDER BY r.FirstName, r.LastName

### Example

|  |  |  |
| --- | --- | --- |
| **FirstName** | **LastName** | **Grade** |
| Agace | Sneddon | 5.99 |
| Anderea | Bowers | 5.99 |
| Andres | Colliard | 5.99 |
| Barbe | Sterrie | 5.75 |
| Baxter | Abrahart | 5.99 |
| … | … | … |

## Not So In The Studying

Find all students **who don’t have any subjects**. Select **their full name**. The full name is combination of first name, middle name and last name. Order the result by **full name**

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

**Solution:**

SELECT CONCAT(s.FirstName + ' ', s.MiddleName + ' ', s.LastName ) AS [Full Name] FROM Students AS s

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

WHERE ss.StudentId IS NULL

ORDER BY [Full Name]

OR

SELECT CONCAT\_WS (' ', s.FirstName, s.MiddleName, s.LastName) AS [Full Name] FROM Students AS s

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

WHERE ss.StudentId IS NULL

ORDER BY [Full Name]

### Example

|  |
| --- |
| **Full Name** |
| Allen Storre Piniur |
| Andria Geleman Andrioletti |
| Ashley Morecombe Summerell |
| Bobby Leggitt Domnin |
| … |

## Top Student per Teacher

Find all teachers with their **top students**. The top student is the person with **highest average grade**. Select teacher full name (first name + last name), subject name, student full name (first name + last name) and corresponding grade. The grade must be formatted to the second digit after the decimal point.

Sort the results by subject name (**ascending**), then by teacher full name (**ascending**), then by grade (**descending**)

### Example

|  |  |  |  |
| --- | --- | --- | --- |
| **Teacher Full Name** | **Subject Name** | **Student Full Name** | **Grade** |
| Farleigh Gerrans | Art | Horatia Kenforth | 5.50 |
| Findlay Collingdon | Art | Zackariah Cordner | 5.27 |
| Ruthanne Bamb | Biology | Merrill Habbijam | 5.75 |
| … | … | … | … |

## Average Grade per Subject

Find the **average** **grade** for each subject. Select the subject name and the average grade.

Sort them by **subject id (ascending)**.

**Solution:**

SELECT F.Name, F.AverageGrade from (SELECT s.Id, s.[Name], AVG(Grade)AS AverageGrade FROM Subjects AS s

JOIN StudentsSubjects AS ss ON s.Id = ss.SubjectId

GROUP BY s.[Name], s.Id ) as F

ORDER BY f.Id

### Example

|  |  |
| --- | --- |
| **Name** | **AverageGrade** |
| Biology | 4.059055 |
| History | 3.880370 |
| English | 4.060546 |
| Math | 3.957876 |
| Music | 3.923984 |
| Art | 4.070898 |
| … | … |

## Exams Information

Divide the year in **4 quarters** using the exam dates. For each quarter get the subject name and the count of students who took the exam with grade more or equal to **4.00**. If the date is missing, replace it with “TBA”. Order them by quarter **ascending**.

**Solution:**

SELECT

CASE

WHEN ISNULL(CAST(DATEPART(QUARTER, [Date]) AS VARCHAR) , 'TBA') = '1' THEN 'Q1'

WHEN ISNULL(CAST(DATEPART(QUARTER, [Date]) AS VARCHAR) , 'TBA') = '2' THEN 'Q2'

WHEN ISNULL(CAST(DATEPART(QUARTER, [Date]) AS VARCHAR) , 'TBA') = '3' THEN 'Q3'

WHEN ISNULL(CAST(DATEPART(QUARTER, [Date]) AS VARCHAR) , 'TBA') = '4' THEN 'Q4'

WHEN ISNULL(CAST(DATEPART(QUARTER, [Date]) AS VARCHAR) , 'TBA') = 'TBA' THEN 'TBA'

END AS [Quarter],

sbj.[Name],

COUNT(sbj.[Name]) AS [Counter]

FROM Exams AS e

JOIN Subjects AS sbj ON e.SubjectId = sbj.Id

JOIN StudentsExams AS se ON e.Id = se.ExamId

WHERE se.Grade >= 4

GROUP BY DATEPART(QUARTER, [Date]) , sbj.[Name]

ORDER BY [Quarter] , sbj.[Name]

### Example

|  |  |  |
| --- | --- | --- |
| **Quarter** | **SubjectName** | **StudentsCount** |
| Q1 | English | 10 |
| Q1 | French | 12 |
| Q1 | Physics | 8 |
| Q2 | English | 10 |
| … | … | … |

# Section 4. Programmability (20 pts)

## Exam Grades

Create a **user defined function**, named **udf\_ExamGradesToUpdate(@studentId, @grade)**, that receives a **student id and grade**.

The function should return the count of grades, for the student with the given id, which are above the received grade and under the received grade with **0.50** added (**example:** you are given grade **3.50** and you have to find all grades for the provided student which are between **3.50** and **4.00** inclusive):

If the condition is true, you must return following message in the format:

* “**You have to update {count} grades for the student {student first name}**”

If the provided student id is not in the database the function should return “The student with provided id does not exist in the school!”

If the provided grade is above **6.00** the function should return “Grade cannot be above 6.00!**”**

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

**Solution:**

CREATE FUNCTION udf\_ExamGradesToUpdate

(@studentId INT, @grade DECIMAL (3,2))

RETURNS NVARCHAR (MAX)

BEGIN

DECLARE @numberOfGrades INT

DECLARE @result NVARCHAR(MAX)

DECLARE @studentName NVARCHAR(50)

IF(@grade > 6.00)

BEGIN

SET @result = 'Grade cannot be above 6.00!'

END

ELSE IF

((SELECT s.Id FROM Students AS s

WHERE s.Id = @studentId) IS NULL )

BEGIN

SET @result = 'The student with provided id does not exist in the school!'

END

ELSE

BEGIN

SET @numberOfGrades = (SELECT COUNT(\*) FROM StudentsSubjects AS ss

WHERE ss.StudentId = @studentId AND ss.Grade BETWEEN @grade + 0.01 AND @grade + 0.50

GROUP BY ss.StudentId)

SET @studentName = (SELECT top(1) s.FirstName FROM StudentsSubjects AS ss

JOIN Students AS s ON ss.StudentId = s.Id

WHERE ss.StudentId = @studentId AND ss.Grade BETWEEN @grade + 0.01 AND @grade + 0.50 )

SET @result = CONCAT('You have to update ', CAST(@numberOfGrades AS NVARCHAR), ' grades for the student ', @studentName )

END

RETURN @result;

END

### Example:

|  |
| --- |
| **Query** |
| **SELECT dbo.udf\_ExamGradesToUpdate(12, 6.20)** |
| **Output** |
| Grade cannot be above 6.00! |

|  |
| --- |
| **Query** |
| **SELECT dbo.udf\_ExamGradesToUpdate(12, 5.50)** |
| **Output** |
| **You have to update 2 grades for the student Agace** |

|  |
| --- |
| **Query** |
| **SELECT dbo.udf\_ExamGradesToUpdate(121, 5.50)** |
| **Output** |
| The student with provided id does not exist in the school! |

## Exclude from school

Create a **user defined stored procedure**, named **usp\_ExcludeFromSchool(@StudentId)**, that receives a **student id** and attempts to **delete the current student**. A student will only be deleted if all of these conditions **pass**:

* If the **student** doesn’t exist, then it **cannot be deleted.** **Raise an error** with the message “This school has no student with the provided id!”

If all the above conditions pass, **delete the student and ALL OF HIS REFERENCES**!

**Solution:**

CREATE PROC usp\_ExcludeFromSchool(@StudentId INT)

AS

BEGIN

IF( (SELECT s.Id FROM Students AS s

WHERE s.Id = @StudentId ) IS NULL)

BEGIN

SELECT 'This school has no student with the provided id!'

END

ELSE

BEGIN

DELETE FROM StudentsTeachers

WHERE StudentId = @StudentId

DELETE FROM StudentsExams

WHERE StudentId = @StudentId

DELETE FROM StudentsSubjects

WHERE StudentId = @StudentId

DELETE FROM Students

WHERE Id = @StudentId

END

END

### Example usage:

|  |  |
| --- | --- |
| **Query** | **Output** |
| **EXEC usp\_ExcludeFromSchool 1**  **SELECT COUNT(\*) FROM Students** | **119** |
| **EXEC usp\_ExcludeFromSchool 301** | This school has no student with the provided id! |