# 

# 

**GOVERNORS STATE UNIVERSITY**

**CPSC-6576-02: Database Design Admin SQL**

**PROJECT**

**STUDENT ACADEMIC DETAILS**

# 

# 

# 

# 

Student Name: Sanjay Boga

Student ID: 1241032

Instructor: Yili Tseng

Table of Contents

[**Introduction** 3](#_Toc102138488)

[**Creating Database** 3](#_Toc102138489)

[**Database Diagram** 4](#_Toc102138490)

[**Normalization** 4](#_Toc102138491)

[**First Normal Form (1NF)** 5](#_Toc102138492)

[**Second Normal Form (2NF) – Functional Dependency** 5](#_Toc102138493)

[**Third Normal Form (3NF) – Transitively Functional Dependency** 6](#_Toc102138494)

[**Implementing and Applying Constraints** 7](#_Toc102138495)

[**Table 1: Course\_Info** 7](#_Toc102138496)

[**Table 2: Student\_Info** 8](#_Toc102138497)

[**Table 3: Marks\_Info** 9](#_Toc102138498)

[**Table 4: Department** 10](#_Toc102138499)

[**Implementing Views** 10](#_Toc102138500)

[**View 1: D01\_Department** 11](#_Toc102138501)

[**View 2: D02\_Department** 11](#_Toc102138502)

[**View 3: D03\_Department** 12](#_Toc102138503)

[**View 4: Student\_CourseWise\_Grading** 12](#_Toc102138504)

[**Implementing Triggers** 13](#_Toc102138505)

[**Create a table for logging the changes** 13](#_Toc102138506)

[**Trigger: Creating an after DML trigger** 14](#_Toc102138507)

[**Testing the trigger (Trigger\_Course\_Info)** 15](#_Toc102138508)

[**Implementing Functions** 16](#_Toc102138509)

[**Fun 1: Get\_StudentDetailsWithPercentage\_Function** 16](#_Toc102138510)

[**Fun 2: Search\_StudentDetailsFunction** 17](#_Toc102138511)

[**Fun 3: CalPercentage\_Function** 18](#_Toc102138512)

[**Implementing Stored Procedures** 19](#_Toc102138513)

[**SP 1: SP\_StudentMarksOperation** 19](#_Toc102138514)

[**SP 2: SP\_FunctionCall\_Search\_StudentDetails** 21](#_Toc102138515)

# **Introduction**

Student academic details is essential for the smooth operation of any university with a large student population. It is a useful database system for university departments because it allows them to store and view information about any student who is currently enrolled in which course. This is built with SQL server, which aids in the management of a university's student list. It also saves individual students' course-by-course grades, among other things.

# **Creating Database**

Creating a database name “**CPSC\_6576\_Student\_Academic\_Details**” for Student academic details to store all the required information using a relational database.

|  |
| --- |
| CREATE DATABASE [CPSC\_6576\_Student\_Academic\_Details] CONTAINMENT = NONE ON PRIMARY ( NAME = N'CPSC\_6576', FILENAME = N'DATA\CPSC\_6576.mdf' ,  SIZE = 8192KB , MAXSIZE = UNLIMITED, FILEGROWTH = 65536KB )  LOG ON  ( NAME = N'CPSC\_6576\_log', FILENAME = DATA\CPSC\_6576\_log.ldf' ,  SIZE = 8192KB , MAXSIZE = 2048GB , FILEGROWTH = 65536KB )  WITH CATALOG\_COLLATION = DATABASE\_DEFAULT GO |

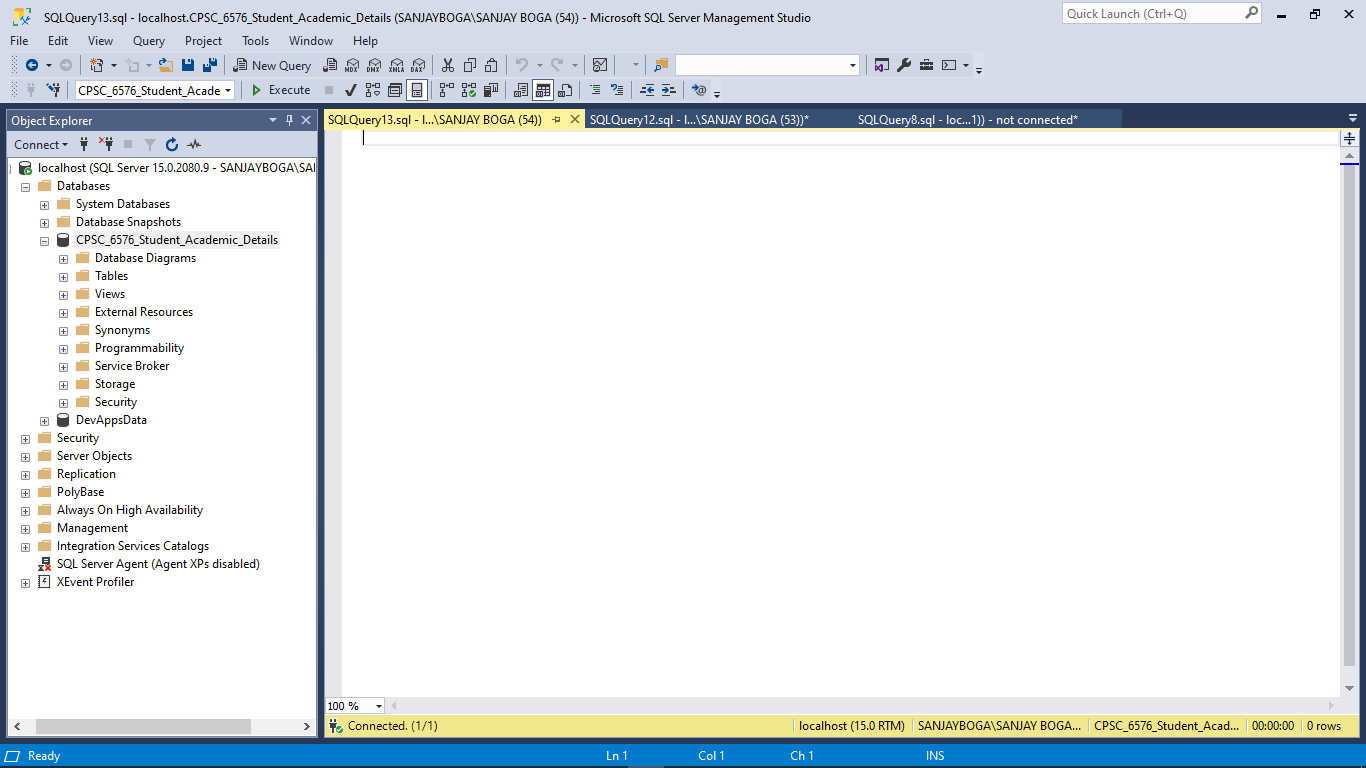


Figure 1: CPSC\_6576\_Student\_Academic\_Details

# **Database Diagram**

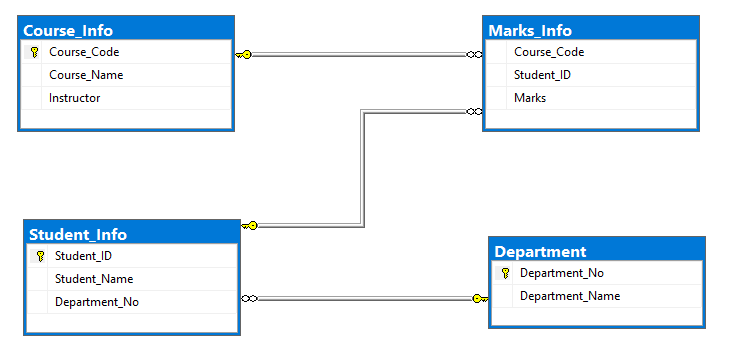


Figure 2: Entity Relationship Diagram: Student Academic Details

ER diagrams facilitate in explaining the logical structure of databases. Entities, attributes, and relationships are the three fundamental concepts upon which ER diagrams are built.

In the above ERD, We have 4 Entities i.e. Course, Student, Marks and Department. The course has 3 attributes which are Course\_Code, Course\_Name and Instructor. A student also has 3 attributes those are Student\_ID, Student Name and Department\_No. Marks Entity consist of 3 attributes that are Course\_Code, Student\_ID and Marks. Finally, Department composes of only 2 attributes i.e. Department\_No and Department\_Name. The diagram above depicts three total relationships: one - to - many between Course Info and Marks Info, Department and Student Info, and Student Info with Marks Info.

# **Normalization**

It, also known as database normalization, is the process of organizing data into database tables. Normalization practices must be followed when creating a good database design. A database system that lacks normalization may even be slow, inefficient, and fail to produce the expected results. Normalization reduces data redundancy and reliance on inconsistent data

To reduce duplication or redundancy of data in **CPSC\_6576\_Student\_Academic\_Details**, we must perform normalisation on the below data.

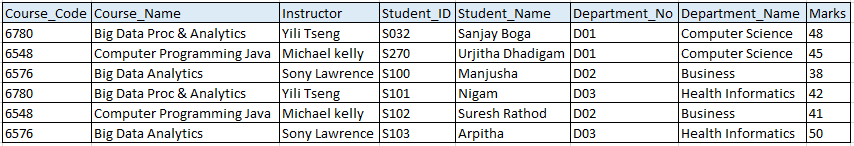
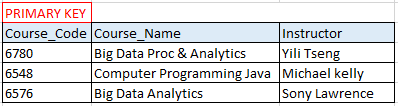


Figure 3: Unnormalized Data

## **First Normal Form (1NF)**

If a database table contains no repeating fields/columns, it is said to be in 1NF.



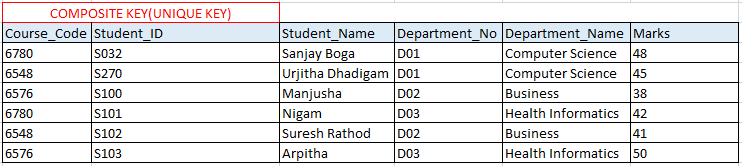
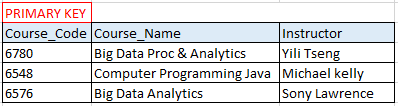
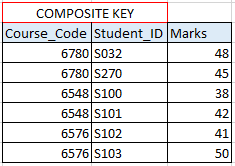


Figure 4: First Normal Form

## **Second Normal Form (2NF) – Functional Dependency**

If a database table is in 1NF and only contains fields/columns that are functionally dependent on the primary key, it is said to be in 2NF.





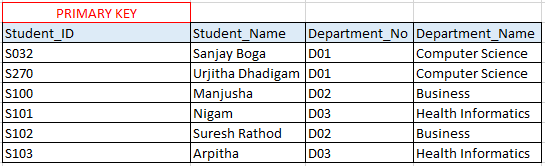


Figure 5: Second Normal Form

## **Third Normal Form (3NF) – Transitively Functional Dependency**

A database table is said to be in 3NF if it is in 2NF and all non-key fields are dependent on the primary key or A table is also said to be in 3NF if it is in 2NF and none of the table's fields is transitively functionally dependent on the primary key.

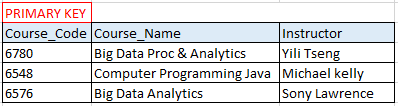


Table 1: Course\_Info

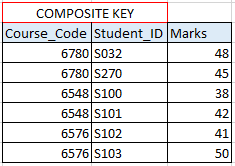


Table 2: Marks\_Info

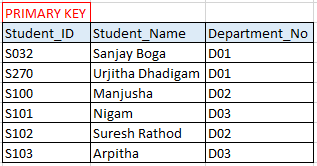
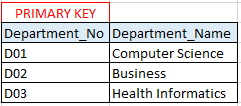
 

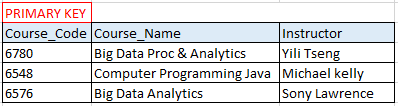
Table 3: Student\_Info Table 4: Department

Figure 6: Third Normal Form

# **Implementing and Applying Constraints**

SQL constraints are used to define the regulations that guide the data in a table. Constraints are used to restrict the types of data that can be entered into a table.

## **Table 1: Course\_Info**



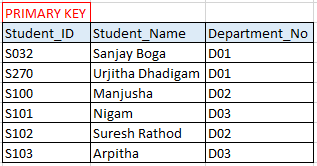
|  |
| --- |
| CREATE TABLE [dbo].[Course\_Info](  [Course\_Code] [int] NOT NULL,  [Course\_Name] [nvarchar](50) NOT NULL,  [Instructor] [nvarchar](50) NOT NULL,  CONSTRAINT [PK\_Course\_Info] PRIMARY KEY CLUSTERED  ( [Course\_Code] ASC )WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY] ) ON [PRIMARY] GO |

|  |
| --- |
| ALTER TABLE [dbo].[Course\_Info] WITH CHECK ADD CONSTRAINT [Check\_Only\_Alphabets] CHECK ((NOT [Course\_Name] like '%[^A-Z ]%')) |

**Constraints Properties**

|  |  |
| --- | --- |
| **Key Constraints** | Course\_Code (PK) UNIQUE , NOT NULL |
| **Entity Key Constraints** | Course\_Code (PK) NOT NULL |
| **Referential Integrity Constraints** |  |
| **Check Constraints** | Course\_Name should only contain alphabets  (NOT [Student\_Name] like '%[^A-Z ]%') |
| **NOT NULL Constraints** | Course\_Code, Course\_Name, Instructor |

## **Table 2: Student\_Info**



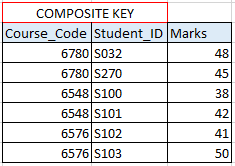
|  |
| --- |
| CREATE TABLE [dbo].[Student\_Info](  [Student\_ID] [nvarchar](10) NOT NULL,  [Student\_Name] [nvarchar](50) NOT NULL,  [Department\_No] [nvarchar](10) NOT NULL,  CONSTRAINT [PK\_Student\_Info] PRIMARY KEY CLUSTERED ( [Student\_ID] ASC )WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY] ) ON [PRIMARY] GO |

|  |
| --- |
| ALTER TABLE [dbo].[Student\_Info] WITH CHECK ADD CONSTRAINT [Check\_Only\_Alphabets\_Student] CHECK ((NOT [Student\_Name] like '%[^A-Z ]%')) |

**Constraints Properties**

|  |  |
| --- | --- |
| **Key Constraints** | Student\_ID (PK) UNIQUE , NOT NULL |
| **Entity Key Constraints** | Student\_ID (PK) NOT NULL |
| **Referential Integrity Constraints** | Department\_No (FK) |
| **Check Constraints** | Student\_Name should only contain alphabets  (NOT [Student\_Name] like '%[^A-Z ]%') |
| **NOT NULL Constraints** | Student\_ID, Student\_Name, Department\_No |

## **Table 3: Marks\_Info**

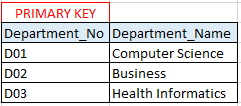


|  |
| --- |
| CREATE TABLE [dbo].[Marks\_Info](  [Course\_Code] [int] NOT NULL,  [Student\_ID] [nvarchar](10) NOT NULL,  [Marks] [int] NOT NULL ) ON [PRIMARY] GO |
| ALTER TABLE [dbo].[Marks\_Info] WITH CHECK ADD CONSTRAINT [Marks] CHECK (([Marks]>(0))) |

**Constraints Properties**

|  |  |
| --- | --- |
| **Key Constraints** | Course\_Code, Student\_ID - UNIQUE , NOT NULL |
| **Referential Integrity Constraints** | Course\_Code (FK), Student\_ID (FK) |
| **Check Constraints** | Marks > 0 (i.e. greater than zero) |
| **NOT NULL Constraints** | Course\_Code, Student\_ID, Marks |

## **Table 4: Department**



|  |
| --- |
| CREATE TABLE [dbo].[Department](  [Department\_No] [nvarchar](10) NOT NULL,  [Department\_Name] [nvarchar](50) NOT NULL,  CONSTRAINT [PK\_Department] PRIMARY KEY CLUSTERED  (  [Department\_No] ASC )WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY] ) ON [PRIMARY] GO |

**Constraints Properties**

|  |  |
| --- | --- |
| **Key Constraints** | Department\_No (PK) - UNIQUE , NOT NULL |
| **Entity Key Constraints** | Department\_No (PK) |
| **NOT NULL Constraints** | Department\_No, Department\_Name |

# **Implementing Views**

A view is a virtual table in SQL that is based on the result set of a SQL statement. Views are used to focus on the concrete columns for the purposes for which they were designed. Views may also be used for security purposes. They filter out columns in the underlying tables that you don't want certain users to see.

In this project based on the data. I have created 4 Views which is used to reduce complexity, and simply respective legacy code.

## **View 1: D01\_Department**

This View holds the data which distinguish data based on Department No i.e. D01

|  |  |  |  |
| --- | --- | --- | --- |
| Student\_ID | Student\_Name | Department\_No | Department\_Name |
| S032 | Sanjay Boga | D01 | Computer Science |
| S270 | Urjitha Dhadigam | D01 | Computer Science |

|  |
| --- |
| CREATE VIEW [dbo].[D02\_Department] AS ( select Student\_ID,Student\_Name,s.Department\_No,Department\_Name  from  Student\_Info s join Department d  on  s.Department\_No = d.Department\_No where d.Department\_No='D02') |

## **View 2: D02\_Department**

This View holds the data which distinguish data based on Department No i.e. D02

|  |  |  |  |
| --- | --- | --- | --- |
| Student\_ID | Student\_Name | Department\_No | Department\_Name |
| S100 | Manjusha | D02 | Business |
| S102 | Suresh Rathod | D02 | Business |
| S998 | D Mahendran | D02 | Business |

|  |
| --- |
| CREATE VIEW [dbo].[D02\_Department] AS ( select Student\_ID,Student\_Name,s.Department\_No,Department\_Name  from  Student\_Info s join Department d  on  s.Department\_No = d.Department\_No where d.Department\_No='D02') |

## **View 3: D03\_Department**

This View holds the data which distinguish data based on Department No i.e. D03

|  |  |  |  |
| --- | --- | --- | --- |
| Student\_ID | Student\_Name | Department\_No | Department\_Name |
| S101 | Nigam | D03 | Health Informatics |
| S103 | Arpitha | D03 | Health Informatics |

|  |
| --- |
| CREATE VIEW [dbo].[D03\_Department] AS ( select Student\_ID,Student\_Name,s.Department\_No,Department\_Name  from Student\_Info s join Department d  on s.Department\_No = d.Department\_No where d.Department\_No='D03') |

## **View 4: Student\_CourseWise\_Grading**

This View holds the data which calculates a grades of student marks based on following condition.

If students score more than 90% the grade is ‘A’, if they score more than 85% than ‘B+’, if they score more than 75% then ‘B-’ else ‘C’

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Student ID | Student\_Name | Course  Code | Course\_Name | Mark | Per% | Grade |
| S032 | Sanjay Boga | 6780 | Big Data Analytics | 48 | 96 | A |
| S270 | Urjitha Dhadigam | 6780 | Big Data Analytics | 45 | 90 | B+ |
| S100 | Manjusha | 6548 | Computer Programming Java | 38 | 76 | B- |
| S101 | Nigam | 6548 | Computer Programming Java | 42 | 84 | B- |
| S102 | Suresh Rathod | 6576 | Database Design Admin SQL | 41 | 82 | B- |
| S103 | Arpitha | 6576 | Database Design Admin SQL | 50 | 100 | A |
| S998 | D Mahendran | 6576 | Database Design Admin SQL | 43 | 86 | B+ |

|  |
| --- |
| CREATE VIEW Student\_CourseWise\_Grading AS( select \*, case when [Per%]>90 then 'A' when [Per%]>85 then 'B+' when [Per%]>75 then 'B-' else 'C' end Grade from ( Select stu.Student\_ID,stu.Student\_Name,c.Course\_Code,c.Course\_Name,marks.Marks,(marks.Marks\*100)/50 'Per%' from  Student\_Info stu join Marks\_Info marks on  stu.Student\_ID = marks.Student\_ID join Course\_Info c on marks.Course\_Code = c.Course\_Code) A) |

# **Implementing Triggers**

A trigger is a type of stored procedure that executes automatically when an event occurs in a database table.

In this project we have 2 triggers

1. **Trigger\_Student\_Info**
2. **Trigger\_Course\_Info**

The above illustrated triggers are used to report the logs of Student\_Info and Course\_Info table based on Insert and Delete operation. So in order to store the logs we have created respective AuditLogs table **Student\_AuditsLogs** and **Course\_AuditsLogs**

## **Create a table for logging the changes**

|  |
| --- |
| CREATE TABLE Course\_AuditsLogs(  Change\_ID INT IDENTITY PRIMARY KEY,  Course\_Code INT NOT NULL,  Course\_Name nvarchar(100) NOT NULL,  Instructor nvarchar(50) NOT NULL,   Updated\_ON DATETIME NOT NULL,  Operation CHAR(3) NOT NULL,  CHECK(Operation = 'INS' or Operation='DEL') ); |

|  |
| --- |
| CREATE TABLE Student\_AuditsLogs(  Change\_ID INT IDENTITY PRIMARY KEY,  Student\_ID INT NOT NULL,  Student\_Name nvarchar(100) NOT NULL,  Department\_No nvarchar(5) NOT NULL,   Updated\_ON DATETIME NOT NULL,  Operation CHAR(3) NOT NULL,  CHECK(Operation = 'INS' or Operation='DEL') ); |

## **Trigger: Creating an after DML trigger**

|  |
| --- |
| CREATE TRIGGER Trigger\_Course\_Info ON [dbo].[Course\_Info] AFTER INSERT, DELETE AS BEGIN  SET NOCOUNT ON;  INSERT INTO [dbo].[Course\_AuditsLogs](  Course\_Code, Course\_Name, Instructor, Updated\_ON,   Operation  )  SELECT  i.Course\_Code, Course\_Name, Instructor, GETDATE(),  'INS'  FROM  inserted i  UNION ALL  SELECT  d.Course\_Code, Course\_Name, Instructor, GETDATE(),  'DEL'  FROM  deleted d; END |

*Trigger\_Course\_Info*

|  |
| --- |
| CREATE TRIGGER Trigger\_Student\_Info ON [dbo].[Student\_Info] AFTER INSERT, DELETE AS BEGIN  SET NOCOUNT ON;  INSERT INTO [dbo].[Student\_AuditsLogs](  Student\_ID, Student\_Name, Department\_No,   Updated\_ON, Operation  )  SELECT i.Student\_ID, Student\_Name, Department\_No,  GETDATE(), 'INS'  FROM  inserted i  UNION ALL  SELECT d.Student\_ID, Student\_Name, Department\_No,  GETDATE(), 'DEL'  FROM  deleted d; END |

*Trigger\_Student\_Info*

## **Testing the trigger (Trigger\_Course\_Info)**

The following statement inserts a new row into the **Course\_Info** table:

|  |
| --- |
| Insert into Course\_Info values  (8810,'Formal Languages and Automata','X Tang'); |

Because of the INSERT event, the **Trigger\_Course\_Info** trigger of **Course\_Info** table was fired.

Let’s examine the contents of the **Course\_AuditLogs** table:

|  |
| --- |
| select \* from [dbo].[Course\_AuditsLogs] |

Here is the output:



# **Implementing Functions**

SQL Server functions are database objects that contain a set of SQL statements to perform a specific task.

Based on our requirements, we can create functions once and use them in multiple locations. The functions for inserting, deleting, or updating records in database tables are not supported by SQL Server.

For our database name “**CPSC\_6576\_Student\_Academic\_Details**” we have created a total of 3 functions in which 2 are table-valued functions which means it returns a single rowset i.e.

## **Fun 1: Get\_StudentDetailsWithPercentage\_Function**

This function is implemented to fetch the student details along with the percentage they gained in a particular course. To calculate percentage we have created a user-defined function which is CalPercentage\_Function a scalar-valued function that returns percentage based on the parameter passed.

|  |
| --- |
| CREATE FUNCTION [dbo].[Get\_StudentDetailsWithPercentage\_Function] () RETURNS TABLE AS RETURN  SELECT s.Student\_ID,s.Student\_Name,c.Course\_Name,m.Marks, [dbo].[CalPercentage\_Function](m.Marks,50) as Percentage  FROM   Student\_Info s  JOIN  Department d  ON  s.Department\_No = d.Department\_No  JOIN  Marks\_Info m  ON  m.Student\_ID=s.Student\_ID  JOIN  Course\_Info c   ON  c.Course\_Code = m.Course\_Code |

Testing above function

|  |
| --- |
| select \* from [dbo].[Get\_StudentDetailsWithPercentage\_Function]() |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Student\_ID | Student\_Name | Course\_Name | Marks | Percentage |
| S032 | Sanjay Boga | Big Data Analytics | 48 | 96 |
| S270 | Urjitha Dhadigam | Big Data Analytics | 45 | 90 |
| S100 | Manjusha | Computer Programming Java | 38 | 76 |
| S101 | Nigam | Computer Programming Java | 42 | 84 |
| S102 | Suresh Rathod | Database Design Admin SQL | 41 | 82 |
| S103 | Arpitha | Database Design Admin SQL | 50 | 100 |
| S998 | D Mahendran | Database Design Admin SQL | 43 | 86 |

## **Fun 2: Search\_StudentDetailsFunction**

This function is implemented to fetch the student details based on Student\_ID as a parameter.

|  |
| --- |
| CREATE FUNCTION [dbo].[Search\_StudentDetailsFunction] (  @Student\_ID NVARCHAR(10) ) RETURNS TABLE AS RETURN  SELECT s.Student\_ID,s.Student\_Name,d.Department\_Name,m.Course\_Code,c.Course\_Name,m.Marks,c.Instructor   FROM   Student\_Info s JOIN Department d ON s.Department\_No = d.Department\_No JOIN Marks\_Info m ON m.Student\_ID=s.Student\_ID JOIN Course\_Info c ON c.Course\_Code = m.Course\_Code  where s.Student\_ID = @Student\_ID |

Testing above function

|  |
| --- |
| select \* from [dbo].[Search\_StudentDetailsFunction]('S032') |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Student\_ID | Student\_Name | Department\_Name | Course\_Code | Course\_Name | Marks | Instructor |
| S032 | Sanjay Boga | Computer Science | 6780 | Big Data Analytics | 48 | Sony Lawrence |

And out which 1 is a scalar-valued function which means it takes one or more values but returns a single value

## **Fun 3: CalPercentage\_Function**

This user-defined function is used to calculate the percentage based on 2 parameters first marks and the second total. CalPercentage\_Function is used as a function within a function.

|  |
| --- |
| CREATE FUNCTION [dbo].[CalPercentage\_Function] (   @MARKS int, @TOTAL INT )  RETURNS int  AS  BEGIN   DECLARE @Result int   SELECT @Result = (@MARKS \* 100) / @TOTAL;     RETURN @Result  END |

Testing above function

|  |
| --- |
| select [dbo].[CalPercentage\_Function](35,50) AS PERCENTAGE |



# **Implementing Stored Procedures**

A SQL stored procedure (SP) is a collection of SQL statements and SQL command logic that is compiled and stored in a database. We can create SQL queries to be stored and executed on the server using stored procedures in SQL. Caching and reusing stored procedures is also possible. The primary goal of stored procedures is to hide direct SQL queries from the code and improve the performance of database operations like select, update, and delete data.

Based on our project requirement we are implementing 2 stored procedures to perform our task.

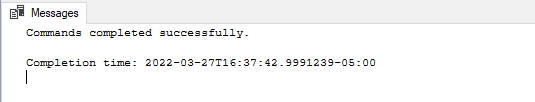
## **SP 1: SP\_StudentMarksOperation**

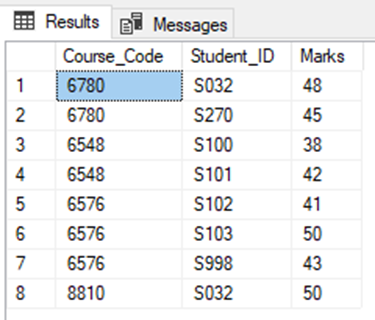
This procedure is designed to perform multiple operations based on the conditional operation. Where end-user tells the procedure to perform types of operation using @viewtype parameter

|  |
| --- |
| CREATE PROCEDURE [dbo].[SP\_StudentsMarksOperation]     @ViewType nvarchar(10) NULL,  @Student\_ID nvarchar(10) NULL,  @Course\_Code int,  @Marks int  AS BEGIN    SET NOCOUNT ON;   IF @ViewType = 'INSERT'  BEGIN   INSERT INTO Marks\_Info VALUES(@Course\_Code,@Student\_ID,@Marks);   END   ELSE IF @ViewType = 'UPDATE'  BEGIN   UPDATE Marks\_Info SET Marks=@Marks WHERE  Course\_Code = @Course\_Code AND Student\_ID = @Student\_ID   END  ELSE IF @ViewType = 'DELETE'  BEGIN  DELETE FROM Marks\_Info WHERE Course\_Code = @Course\_Code   AND Student\_ID = @Student\_ID  END END |

Executing Above Stored Procedure:

|  |
| --- |
| EXEC [dbo].[SP\_StudentsMarksOperation]  @ViewType='INSERT',@Student\_ID='S032',@Course\_Code='8810',@Marks=50 |





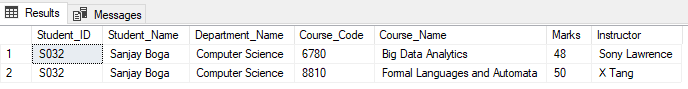
## **SP 2: SP\_FunctionCall\_Search\_StudentDetails**

In this Stored Procedure we are calling the user-defined function which we created earlier in this project name Search\_StudentDetailsFunction which takes one parameter as input to fetch records based on it.

|  |
| --- |
| CREATE PROCEDURE SP\_FunctionCall\_Search\_StudentDetails   @Student\_ID nvarchar(10) NULL AS BEGIN   SET NOCOUNT ON;   SELECT \* from [dbo].[Search\_StudentDetailsFunction]  (@Student\_ID); END GO |

Testing above Stored Procedure:

|  |
| --- |
| EXEC [dbo].[SP\_FunctionCall\_Search\_StudentDetails] @Student\_ID = 'S032' |



* END -