# Views

Views are also called *virtual* or *derived tables* because they present the user with what appear to be tables; however, the information in those tables is derived from previously defined table.

# Example of a View:

Create View StudentAndCourseInfo

As

Select Students.rollNumber as "Roll Number", Students.name as "Name", Courses.name as "Course Name"

From Students Join StudentCourses on Students.rollNumber=StudentCourses.rNo Join Courses On StudentCourses.cid=Courses.cid

Or

Create View StudentAndCourseInfo ("Roll Number", Name, "Course Name")

As

Select Students.rollNumber, Students.name, Courses.name

From Students Join StudentCourses on Students.rollNumber=StudentCourses.rNo Join Courses On StudentCourses.cid=Courses.cid

# Using the View:

Select \* From StudentAndCourseInfo

You can also use where statement with views.

Select \* From StudentAndCourseInfo where name='abc'

# Benefits of Views:

1. Can restrict data access.
2. Can reduce network traffic.
3. Can reduce the complexity of query.

# Materialized Views:

By default, a view is a virtual table, i.e., it does not actually contain data. The data comes at run time from the base table. To improve performance, we can create the view as materialized view which will have the result as well. The benefits of materialized views are that results will be produced faster. The drawbacks are: storage requirements and update requirements when the base table(s) is/are changed.

# Update Strategies for Materialized Views

Different strategies as to when a materialized view is updated are possible.

1. The **immediate update** strategy updates a view as soon as the base tables are changed.
2. The **lazy update** strategy updates the view when needed by a view query.
3. The **periodic update** strategy updates the view periodically (in the latter strategy, a view query may get a result that is not up-to-date).

# Stored Procedures

Stored Procedure in SQL server can be defined as the set of logically group of SQL statement which are grouped to perform a specific task. There are many benefits of using a stored procedure.

A stored procedure is nothing more than a prepared SQL code that you save so that you can reuse the code over and over again.

Example:

|  |  |
| --- | --- |
| **Benefit** | **Explanation** |
| Modular Programming | * You can write a stored procedure once, then call it from multiple places in your application hence reducing development time * It can accept input parameters, return output values as parameters, or return success or failure status messages |
| Performance | * Stored procedures provide faster code execution * Reduced network traffic |
| Security | * Users can execute a stored procedure without needing to execute any of the statements directly * Users can specifically be granted permission to execute only stored procedures instead of allowing them to execute queries on tables directly. * They prevent SQL injection. |
| Separation B/W Data Logic and Application. | * No need to write SQL queries in the application itself since the query will be stored in the form of a stored procedure in the database. In the application, you only need to call the procedure (from the database) using its name. |

Every time you execute and SQL statements syntax Check, Compilation and done before Execution and Return data.

However Syntax check and Compilation is done while creating a procedure, and not on every execution which makes in faster than simple SQL statements.

## Variables.

Before we start with stored procedures, we should get to know the variables. Like in any other programing language SQL also provides scalar variables, which are very useful when creating stored procedures.

* Variable in SQL start with @ symbol
* Variable is declared using DECLARE keyword as follow
  + *DECLARE @variableName datatype;*

Or to declare multiple variables in one statement.

* + *DECLARE @variable1Name Datatype,@variable2Name  datatype;*
* Variable can be assigned a constant scalar value as follow
  + *SET  @ variableName  = value;*

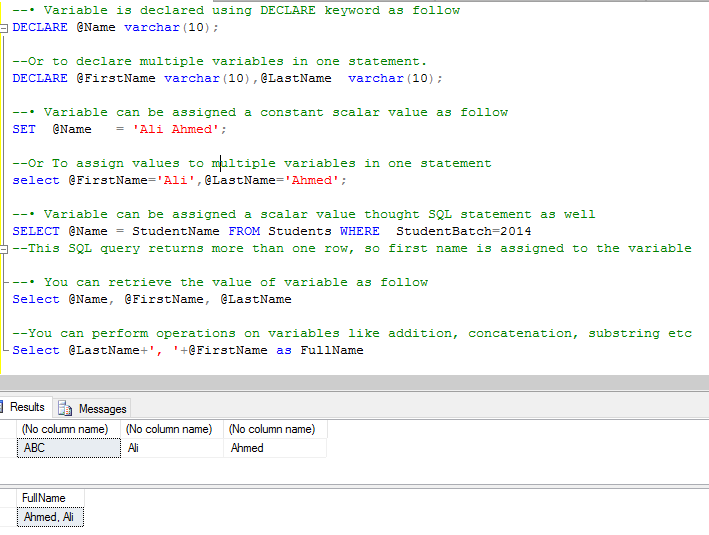
Or To assign values to multiple variables in one statement

* + *select @ variable1Name   = value, @variable2Name  =value;*
* Variable can be assigned a scalar value thought SQL statement as well
  + *SELECT @vairableName = columnName FROM Table WHERE  <condition>*

If SQL query returns more than one row, 1st value will be assigned to variable

* You can retrieve the value of variable as follow
  + *Select @variableName*
* You can perform operations on variables like addition, concatenation, substring etc

TRY IT



NOTE: USE AND DECLARE VARIABLE IN SAME BATCH OF STATEMENTS, IF DECLARE STATEMENT IS NOT IN SAME BATCH, YOU WILL ERRON WHILE USING A VARIABLE.

## CREATE Stored Procedure

Following is the syntax to create stored procedure: Input and output parameter a uses as required.

CREATE PROCEDURE [procedureName]

@input\_param1 datatype,

@input\_param2 datatype,

@output\_param1 datatype OUTPUT,

@output \_param2 datatype OUTPUT

AS

BEGIN

(SQL Queries)

END

go

## How to execute Stored Procedure

declare @my\_output\_param1 int,

@my\_output\_param2 varchar(10) --these are the variables in which output variables of procedure will return values

Exec dbo.procedure\_name

@input\_param1=value,

@input\_param2 =value,

@output\_param1=@my\_output\_param1 OUTPUT ,

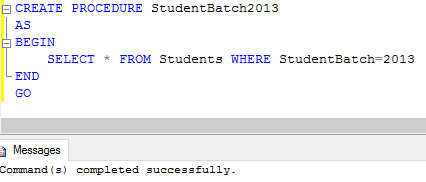
@output\_param2 =@my\_output\_param2 OUTPUT

select @my\_output\_param1 ,@my\_output\_param2 – you will then have to use select statements to retrieve data from parameters

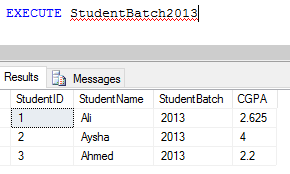
### Stored Procedures without I/O parameters

TRY IT:

Create this procedure to obtain all the students of batch 2013



Now execute this procedure

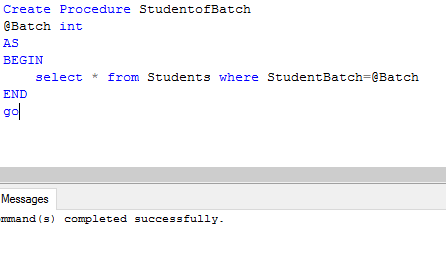


### 

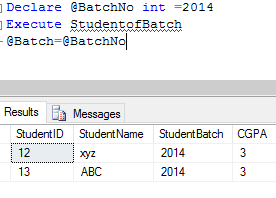
### Stored procedure with input parameters

TRY IT

Create a SP which takes batchNo as input and returns all students of that batch



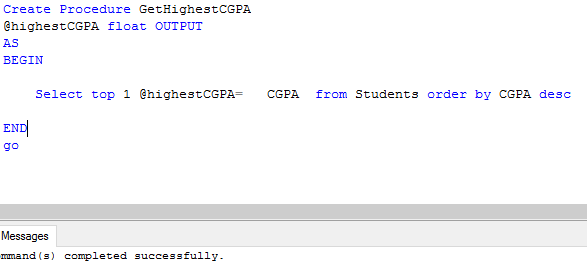
Now execute it



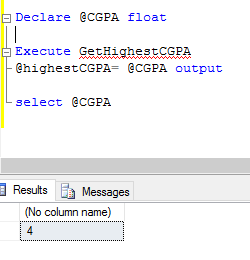
### Store Procedures with output parameters

TRY IT:

Create a stored procedure that will return max CGPA in an output parameter



Execute it

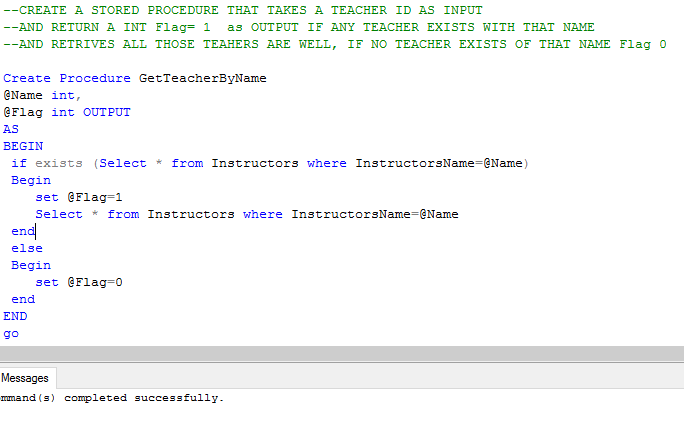


QUESTION: WRITE A SP TO GET AVERAGE CGPA.

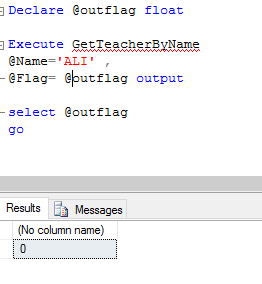
### IF-ELSE conditions

Like in any programing language IF—ELSE in SQL provide ability to conditionally execute a code.

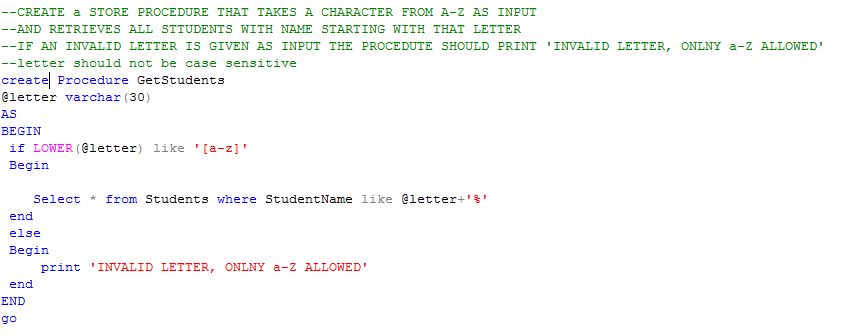
TRY THIS



Execute it



TRY ANOTHER



TRY EXECUTING THESE

execute GetStudents @letter= 'B'

execute GetStudents @letter= '1'