ADVANTAGES OF STORE PROCEDURES

1. **Execution plan retention and reusability:** Stored Procedures are compiled and their execution plan is cached and used again, when the same SP is executed again. Although ad-hoc queries also create and reuse plan, the plan is reused only when the query is textual match and the data types are matching with the previous call. Any change in the data type or you have an extra space in the query then, a new plan is created.

2. **Reduces network traffic:** You only need to send, EXECUTE Stored Procedure Name statement, over the network, instead of the entire batch of ad-hoc SQL code.

3. **Code reusability and better maintainability:** A stored procedure can be reused with multiple applications. If the logic has to change, we only have one place to change, where as if it is inline SQL, and if you have to use it in multiple applications, we end up with multiple copies of this inline SQL. If the logic has to change, we have to change at all the places, which makes it harder maintaining inline SQL.

4. **Better Security:** A database user can be granted access to an SP and prevent them from executing direct "select" statements against a table. This is fine grain access control which will help control what data a user has access to.

5. **. Avoids SQL Injection attack:** Stored Procedure prevent SQL injection attack.

What is SQL Injection?

- SQL injection is a code injection technique that might destroy your database.
- SQL injection is one of the most common web hacking techniques.
- SQL injection is the placement of malicious code in SQL statements, via web page input.

https://www.w3schools.com/sql/sql_ref_mysql.asp
This link is for more built-in string functions.

For Example:

SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will **unknowingly** run on your database.

Look at the following example which creates a SELECT statement by adding a variable (txtUserId) to a select string. The variable is fetched from user input (getRequestString):

PHP Code:

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

Practical Example 1: (SQL Injection Based on 1=1 is Always True)

Look at the example above again. The original purpose of the code was to create an SQL statement to select a user, with a given user id.

If there is nothing to prevent a user from entering "wrong" input, the user can enter some "smart" input like this:

UserId: | 105 OR 1=1 |

Then, the SQL statement will look like this:

```
SELECT * FROM Users WHERE UserId = 105 OR 1=1;
```

The SQL above is valid and will return ALL rows from the "Users" table, since **OR 1=1** is always TRUE.

Does the example above look dangerous? What if the "Users" table contains names and passwords?

A hacker might get access to all the user names and passwords in a database, by simply inserting 105 OR 1=1 into the input field.

https://www.w3schools.com/sql/sql_ref_mysql.asp
This link is for more built-in string functions.

Practical Example 2: (SQL Injection Based on Batched SQL Statements)

Most databases support batched SQL statement.

A batch of SQL statements is a group of two or more SQL statements, separated by semicolons.

The SQL statement below will return all rows from the "Users" table, then delete the "Suppliers" table.

SELECT * FROM Users; DROP TABLE Suppliers

Look at the following example:

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

And the following input:

UserId: | 105; Drop Table |

The valid SQL statement would look like this:

SELECT * FROM Users WHERE UserId = 105; DROP TABLE Suppliers;

So it can drop the suppliers table from your database unknowingly.

How to get protection from SQL Injection?

- To protect a web site from SQL injection, you can use SQL parameters.
- SQL parameters are values that are added to an SQL query at execution time, in a controlled manner.
- Note that parameters are represented in the SQL statement by a @ marker.
- The SQL engine checks each parameter to ensure that it is correct for its column and are treated literally, and not as part of the SQL to be executed.

https://www.w3schools.com/sql/sql_ref_mysql.asp
This link is for more built-in string functions.

BUILT-IN STRING FUNCTION

1. **ASCII(Character_Expression):** Returns the ASCII code of the given character expression. To find the ACII Code of capital letter 'A'

   Example: Select ASCII('A')
   Output: 65

2. **CHAR(Integer_Expression):** Converts an int ASCII code to a character. The Integer_Expression, should be between 0 and 255.

   Example: Select CHAR(65);
   Output: A

3. **LTRIM(Character_Expression):** Removes blanks on the left handside of the given character expression.

   Example: Removing the 3 white spaces on the left hand side of the '   Hello' string using LTRIM() function.
   Select LTRIM('   Hello')
   Output: Hello

4. **RTRIM(Character_Expression):** Removes blanks on the right hand side of the given character expression.

   Example: Removing the 3 white spaces on the left hand side of the 'Hello   ' string using RTRIM() function.
   Select RTRIM('Hello   ')
   Output: Hello

5. To remove white spaces on either sides of the given character expression, use LTRIM() and RTRIM() as shown below.

   Example: Select LTRIM(RTRIM('   Hello   '))
   Output: Hello

6. **LOWER(Character_Expression):** Converts all the characters in the given Character_Expression, to lowercase letters.

https://www.w3schools.com/sql/sql_ref_mysql.asp
This link is for more built-in string functions.

Example: Select LOWER('CONVERT This String Into Lower Case')
Output: convert this string into lower case

7. **UPPER(Character_Expression):** Converts all the characters in the given Character_Expression, to uppercase letters.

   Example: Select UPPER('CONVERT This String Into upper Case')
   Output: CONVERT THIS STRING INTO UPPER CASE

8. **REVERSE('Any_String_Expression'):** Reverses all the characters in the given string expression.

   Example: Select REVERSE('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
   Output: ZYXWVUTSRQPONMLKJIHGFEDCBA

9. **LENGTH(String_Expression):** Returns the count of total characters, in the given string expression, excluding the blanks at the end of the expression.

   Example: Select LENGTH('SQL Functions ')
   Output: 13

10. **LEFT(Character_Expression, Integer_Expression):** Returns the specified number of characters from the left hand side of the given character expression.

    Example: Select LEFT('ABCDE', 3)
    Output: ABC

11. **RIGHT(Character_Expression, Integer_Expression):** Returns the specified number of characters from the right hand side of the given character expression.

    Example: Select RIGHT('ABCDE', 3)
    Output: CDE

12. **SUBSTRING('Expression', 'Start', 'Length'):** As the name, suggests, this function returns substring (part of the string), from the given expression. You specify the starting location using the 'start' parameter and the number of characters in the substring using 'Length' parameter. All the 3 parameters are mandatory.

    Example: Display just the domain part of the given email 'John@bbb.com'.
    Select SUBSTRING('zubair@bbb.com',6, 7)
    Output: bbb.com

https://www.w3schools.com/sql/sql_ref_mysql.asp
This link is for more built-in string functions.