# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**"Jnana Sangama", Belagavi-560014**

**DBMS Seminar Report On**

*"Exploring the Stored Procedures and SQL"*

**Submitted in partial fulfillment of the requirement of V Semester DBMS**

Submitted by


| | |
|---|---|
| **PRAJWAL P** | **1DT21IS111** |
| **PRAJWAL S GOWDA** | **1DT21IS112** |
| **PRAJWAL T P** | **1DT21IS113** |
| **PREETHAM S** | **1DT21IS114** |
| **PUNEETH P** | **1DT21IS115** |


**Under the guidance of**


**Mrs. Gagana B R**
Asst. Professor
Dept. of ISE
DSATM, Bangalore.


**DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING**

**DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT**

(Affiliated to Visvesvaraya Technological University, Belagavi & Approved by AICTE, New Delhi)
(All B.E Courses Accredited by NBA, New Delhi)
Opp. Art of Living, Udayapura, Kanakapura Road, Bangalore- 560082 **2023-24**

# DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT

**(Affiliated to Visvesvaraya Technological University, Belagavi & Approved by AICTE, New Delhi)**
(All B.E Courses Accredited by NBA, New Delhi)
**Opp. Art of Living, Udayapura, Kanakapura Road, Bangalore- 560082**

## DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING


# *Certificate*

This is to certify that the seminar work entitled **"Exploring the Stored Procedures and SQL"** is carried out by **PRAJWAL P(1DT21IS111), PRAJWAL S GOWDA (1DT21IS112), PRAJWAL T P (1DT21IS113), PREETHAM S (1DT21IS114), PUNEETH P(1DT21IS115)** in partial fulfillment for the requirement of V semester DBMS in **Information Science and Engineering** of the **Visvesvaraya Technological University, Belagavi** during the year 2023-2024. It is certified that all the corrections/ suggestions indicated for the given internal assessment have been incorporated in the report. This report has been approved as it satisfies the academic requirements with respect to the mini-project work.

**Signature of the Guide**

**Mrs. Gagana B R**
Asst. Professor, Dept. of ISE
DSATM, Bangalore.

**Signature of the HOD**

**Dr. Nandini Prasad**
Dean (Foreign Affairs) and HOD
Dept. of ISE
DSATM, Bangalore.

# ACKNOWLEDGEMENT

We express our profound gratitude to **Dr. Ravishankar. M, Principal, DSATM, Bangalore,** for providing the necessary facilities and an ambient environment to work.

We are grateful to **Dr. Nandini Prasad K S, Dean (Foreign Affairs) & HOD, Department of Information Science and Engineering, DSATM, Bangalore,** for her valuable suggestions and advice throughout my/our work period.

We would like to express my deepest gratitude and sincere thanks to our guide **Mrs. Gagana B R, Assistant Professor, Department of Information Science and Engineering, DSATM, Bangalore,** for her keen interest and encouragement in the project whose guidance made the project into reality. We would like to thank all the staff members of **Department of Information Science and Engineering** for their support and encouragement during the course of this project.

Definitely most, We would like to thank our parents, all my family members and friends, without whose help and encouragement this project would have been impossible.

PRAJWAL P (1DT21IS111)
PRAJWAL S GOWDA (1DT21IS112)
PRAJWAL T P (1DT21IS113)
PREETHAM S (1DT21IS114)
PUNEETH P(1DT21IS115)

# ABSTRACT

Stored procedures are a fundamental aspect of database management systems, offering a structured approach to executing SQL statements. Their purpose extends beyond mere data manipulation, serving as a means to encapsulate complex logic and procedural operations within the database environment. The syntax for creating stored procedures may vary slightly between different database systems, but generally includes sections for declaration and implementation, often supporting parameters for dynamic execution. One of the significant advantages of stored procedures lies in their ability to enhance security by controlling access to database objects and reducing the risk of unauthorized access. Additionally, they contribute to improved performance by minimizing network traffic and promoting code reusability. Maintenance tasks are streamlined as stored procedures centralize logic, facilitating updates and ensuring consistency across applications. Debugging and testing procedures are simplified, as stored procedures can be isolated and examined independently. Despite differences in syntax, the concept of stored procedures is widely supported across various database systems, making them a portable and indispensable tool for efficient database management.

# TABLE OF CONTENTS

# CHAPTER 1

## STORED PROCEDURES

Stored procedures represent a cornerstone of modern database management systems, offering a robust mechanism for encapsulating and executing sets of SQL statements. Essentially, a stored procedure is a precompiled collection of SQL commands stored within the database system itself. This abstraction allows for the grouping of SQL statements into a single unit, which can then be invoked by applications or other procedures.

The primary purpose of stored procedures is to provide a centralized location for frequently used or complex database operations. By encapsulating logic within a stored procedure, developers can promote code reuse, simplify maintenance, and enhance security. Stored procedures can contain a wide range of SQL commands, including data manipulation (such as INSERT, UPDATE, DELETE), data retrieval (SELECT), and procedural logic (IF-ELSE statements, loops, etc.).

One of the key benefits of stored procedures is their ability to improve performance. Because stored procedures are precompiled and stored in the database, they can be executed more efficiently than ad-hoc SQL queries sent from client applications. This efficiency is particularly noticeable in scenarios where complex operations need to be performed repeatedly, as the overhead of parsing and compiling SQL statements is eliminated.

Stored procedures also offer a layer of security by controlling access to database objects. Instead of granting direct access to tables or views, permissions can be granted to execute specific stored procedures. This helps to limit potential security vulnerabilities and ensure that database operations are performed in a controlled and consistent manner.

Furthermore, stored procedures enable transaction management within the database environment. Multiple SQL statements can be grouped together within a single stored procedure, allowing for the execution of atomic transactions that ensure data integrity and consistency.

# CHAPTER 2

## CREATING A SIMPLE STORED PROCEDURES

**1**.**CREATE PROCEDURE**: This statement is used to define a new stored procedure in the database. It's followed by the name of the procedure you want to create

**2**.**Procedure Name**: This is the identifier for your stored procedure. Choose a name that is descriptive of the task or operation the procedure will perform. In our example, we named it `GetEmployee` to indicate that it retrieves employee information.

**3**. **Parameters**: Parameters are placeholders for values that are passed into the stored procedure when it is executed. They enable flexibility and reusability by allowing you to customize the behaviour of the procedure without modifying its code. In our example, `@EmployeeID INT` declares an input parameter named `@EmployeeID` of type `INT`, which will be used to specify the employee ID.

**4. AS BEGIN ... END**: This block encapsulates the body of the stored procedure. Anything placed between `AS` and `BEGIN` is considered the declaration section, where you define the parameters and any local variables. Everything between `BEGIN` and `END` is the procedural logic, consisting of SQL statements that perform the desired task.

**5. SQL Statements**: Within the `BEGIN ... END` block, you write the SQL statements that constitute the functionality of the stored procedure. These statements can include any valid SQL commands supported by your database system. In our example, we used a `SELECT` statement to retrieve employee details from the `employees` table based on the provided `EmployeeID`.

**6. Parameter Usage**: Inside the procedural logic, you can reference the input parameters declared in the procedure definition. This allows you to use the parameter values dynamically within your SQL statements. In our example, `@EmployeeID` is used in the `SELECT` query to filter the results based on the provided employee ID

.

# CHAPTER 3

## CALLING A STORED PROCEDURE

**1. EXEC keyword**: In SQL, the `EXEC` keyword is used to execute a stored procedure. It's followed by the name of the stored procedure you want to call.

**2**. **Stored Procedure Name**: After the `EXEC` keyword, you specify the name of the stored procedure you want to call. In our example, we're calling the `GetEmployee` stored procedure.

**3. Parameters**: If the stored procedure has parameters defined, you need to provide values for those parameters when calling the procedure. In our example, `@EmployeeID = 123` specifies the value `123` to be passed to the `@EmployeeID` parameter of the `GetEmployee` stored procedure.

**4**. **Parameter Passing**: There are different ways to pass parameters to a stored procedure, depending on the database system you're using. In our example, we're using named parameter syntax, where we explicitly specify the parameter name (`@EmployeeID`) followed by the value (`123`). This method makes the code more readable and less error-prone, especially when dealing with procedures with multiple parameters.

**5**. **Execution**: When you execute the SQL statement containing the `EXEC` command, the **specified stored procedure is invoked with the provided parameters. The stored procedure's logic is** executed, and any result or side effects (such as database updates) are produced as a result of the procedure's execution.

**6**. **Handling Results**: Depending on the stored procedure's logic, it may produce output such as query results, affected row counts, or other indicators. These results can be handled in various ways, depending on the application's requirements, such as capturing the results in variables or processing them directly.

# CHAPTER 4

## SQL/PSM

Strict SQL/PSM stands for SQL/Procedural Standard Language. It's an extension to SQL (Structured Query Language) that adds procedural programming language features. SQL/PSM allows developers to write procedural code directly within the database, enabling tasks such as looping, conditional execution, and exception handling, which are not directly supported by standard SQL.

**1. Procedural Constructs**: SQL/PSM introduces procedural constructs such as loops (`WHILE`, `FOR`), conditional statements (`IF`, `CASE`), and exception handling (`BEGIN...END` blocks with `TRY...CATCH`).

**2. Stored Procedures and Functions**: With SQL/PSM, you can define stored procedures and functions containing procedural code. Stored procedures are reusable blocks of code stored in the database that can be invoked by applications. Functions return a single value and can be used within SQL queries.

**3. Variables and Control Flow SQL/PSM** allows the declaration and use of variables within procedural code. These variables can hold data values that can be manipulated during the execution of the procedure. Control flow statements such as `IF`, `WHILE`, and `FOR` enable developers to implement conditional and iterative logic.

**4. Exception Handling:** SQL/PSM provides mechanisms for handling errors and exceptions within procedural code. This includes `TRY...CATCH` blocks, which allow developers to catch and handle exceptions gracefully.

**5. Transaction Management**: SQL/PSM supports transaction management, allowing developers to define explicit transactions within stored procedures. This enables developers to control the boundaries of transactions and ensure data integrity.

**6. Enhanced Functionalit**y: By combining SQL/PSM with standard SQL, developers can create powerful and complex database applications. SQL/PSM enables developers to implement business logic directly within the database, improving performance and maintainability.

## 4.1 ADVANTAGES

**1. Procedural Logic**: SQL/PSM allows developers to implement procedural logic directly within the database. This enables the creation of complex business rules, data transformations, and data validation procedures, enhancing the functionality of database applications.

**2. Code Reusability**: Stored procedures and functions created with SQL/PSM can be reused across multiple database applications. This promotes code modularity and reduces duplication of code, leading to improved maintainability and efficiency.

**3. Improved Performance**: By moving business logic to the database layer, SQL/PSM can improve performance by reducing network traffic and minimizing data transfer between the database server and client applications. This is particularly beneficial for operations that involve large datasets or complex calculations.

**4. Enhanced Security**: SQL/PSM allows developers to encapsulate sensitive operations within stored procedures, limiting direct access to underlying data tables. This enhances security by enforcing access control and reducing the risk of SQL injection attacks.

**5. Transaction Management**: SQL/PSM provides robust transaction management capabilities, allowing developers to define explicit transaction boundaries within stored procedures. This ensures data consistency and integrity by grouping related database operations into atomic units of work.

## 4.2 DISADVANTAGES:

**1. Complexity**: Implementing procedural logic within the database can introduce complexity, especially for developers who are primarily familiar with SQL. Writing and maintaining complex stored procedures requires a solid understanding of both SQL and procedural programming concepts.

**2. Vendor Specificity**: Although SQL/PSM is part of the SQL standard, the specific syntax and features may vary between different database vendors. This can lead to portability issues when migrating stored procedures between different database platforms.

**3. Debugging and Testing**: Debugging and testing stored procedures can be challenging compared to traditional SQL queries. Testing procedural code requires specialized tools and techniques, and debugging stored procedures often involves logging and tracing mechanisms provided by the database system.

**4. Performance Bottlenecks**: While SQL/PSM can improve performance in many scenarios, poorly optimized stored procedures can introduce performance bottlenecks. Inefficient procedural code, excessive resource consumption, and improper indexing can degrade database performance and scalability.

**5. Maintenance Overhead**: As stored procedures become more complex and proliferate within a database system, managing and maintaining them can become challenging. Changes to business requirements may necessitate frequent updates to stored procedures, increasing maintenance overhead and potentially introducing regression issues.

.

# CHAPTER 5

# CONCLUSION

In summary, stored procedures serve as powerful tools within database systems, offering a range of advantages such as enhanced security, improved performance, and streamlined maintenance. By encapsulating database logic, they promote code reusability and facilitate efficient data management. However, their implementation requires careful consideration of factors like complexity and performance optimization. Nonetheless, stored procedures remain indispensable for developing robust, scalable, and secure database applications, making them a cornerstone of modern database development practices.

.