

INFSCI 1022

Database Management Systems

Variables, Stored Procedures, Views, Functions

Today's Evil Plan

- Learn about variables, functions, stored procedures, views

Variables

Views

Functions

Stored Procedures

Triggers

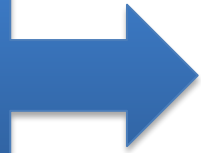
Variables in Math

$$x = 5$$

$$c^2 = a^2 + b^2$$

Variables in Java

int indicates (or sets) the type of variable



```
int x = 5;
```

```
int x = 5;
```

This code creates a **variable** that is of type **integer** and has a **value** of 5

Variables in Python

Note that the
data type is not
specified




```
x = 5
```



```
x = 5;
```


This code creates a **variable** that would be **interpreted** as of type **integer** and has a **value** of 5

$x = 5;$




Value of variable
“x” is “5”

$x = 7;$



Now, value of
variable “x” is “7”

$x = x * 2;$



What is the value
of “x” now?

Variables

Variables in SQL

Control Flow Functions

Commit

Transaction

Rollback

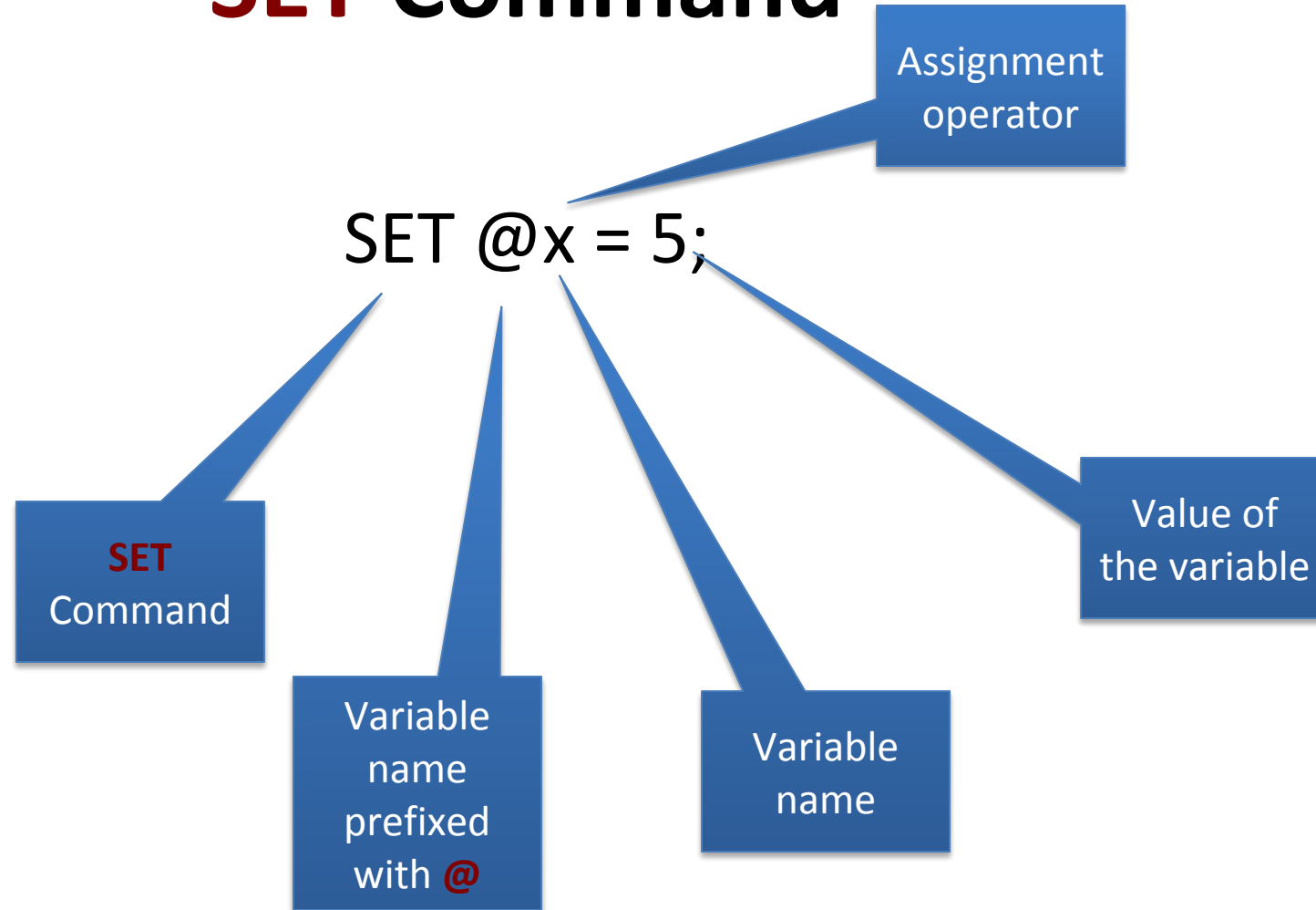
SET Command

```
SET @x = 5;
```

SET Command

- You can create variables in SQL using the **SET** command
- Variable names must be prefixed with **@**
- Variable names can contain any combination of numbers and letters
- The only special character that is allowed is underscore (**_**)

SET Command



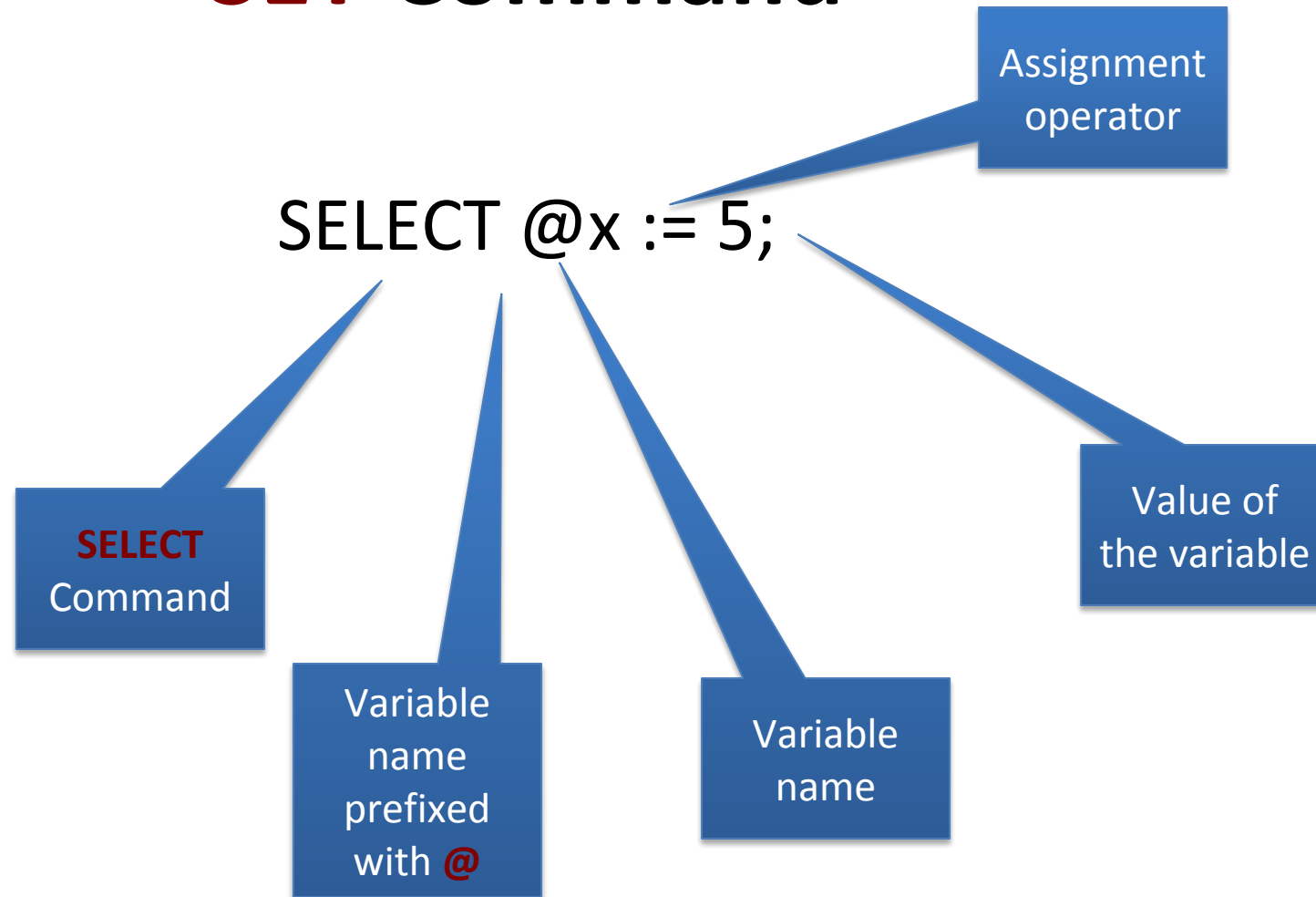
SELECT Command

```
SELECT @x := 5;
```

SELECT Command

- You can create variables in SQL using the **SELECT** command
- Variable names must be prefixed with **@**
- Variable names can contain any combination of numbers and characters
- The only special character that is allowed is underscore (**_**)

SET Command



Assignment vs. Comparison

- Comparison operator =
- Assignment operator :=
- SELECT interprets = (equal sign) as a comparison operator
 - Ex: SELECT * FROM employees WHERE employee_id = 5;
- To be on the safe side, use := for variable value assignments

Everything about SQL Variables

<http://dev.mysql.com/doc/refman/5.0/en/user-variables.html>

Variables

Views

Functions

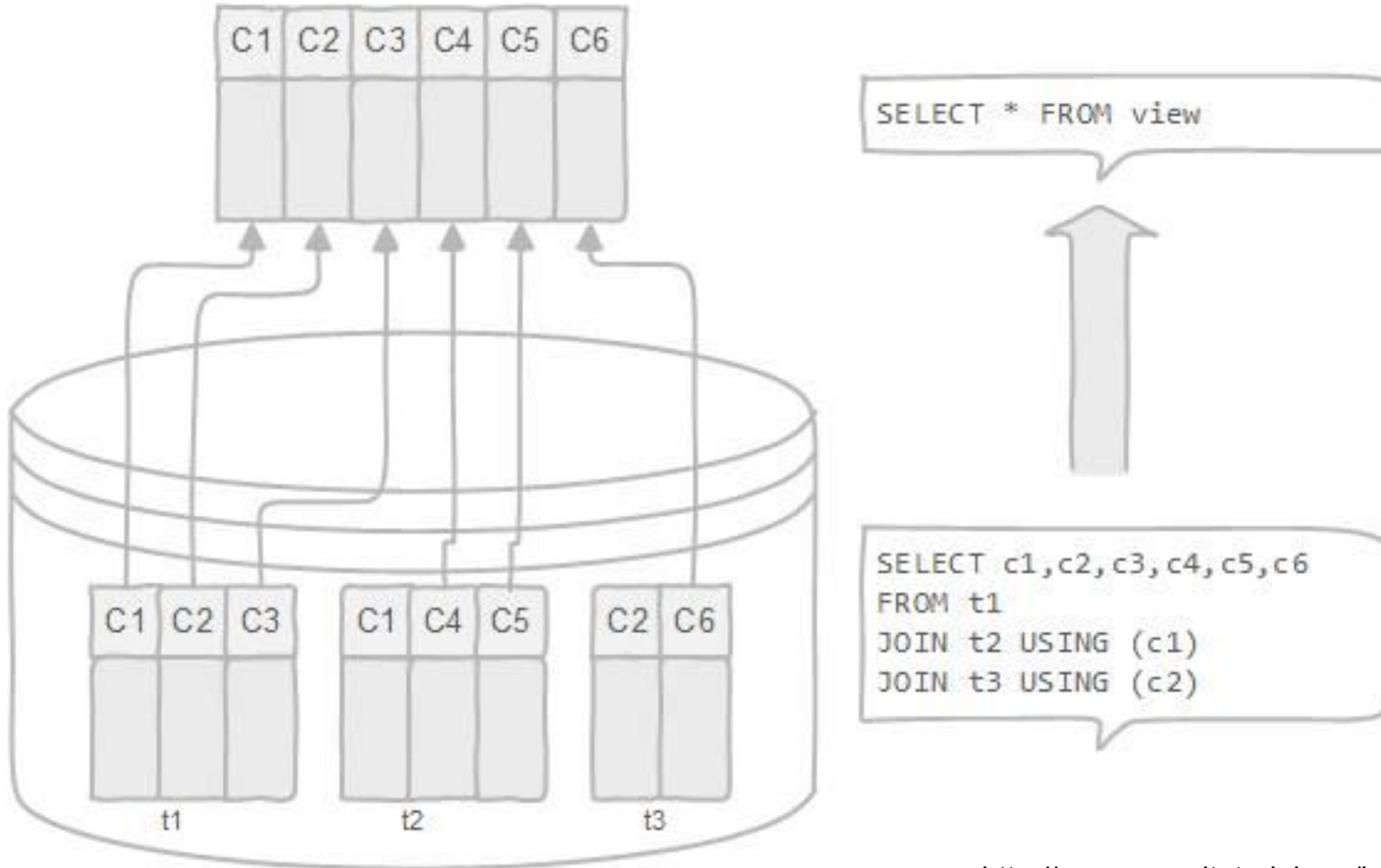
Stored Procedures

Triggers

Introduction to Views

- A database view is a virtual table or logical table which is defined as a SQL SELECT query with joins
- Because a database view is similar to a database table, which consists of rows and columns, so you can query data against it
- Most DBMS allow you to update data in the underlying tables through the database view with some prerequisites

Introduction to Views



Introduction to Views

- A database view is dynamic because it is not related to the physical schema.
- The database system stores views as a SQL SELECT statement with joins.
- When the data of the tables changes, the view reflects that changes as well.

Views: Advantages

Allows you to simplify complex queries

- A database view is defined by an SQL statement that associates with many underlying tables
- You can use database view to hide the complexity of underlying tables to the end-users and external applications.
- Through a database view, you only have to use simple SQL statements instead of complex ones with many joins.

Views: Advantages

- Helps limit data access to specific users.
 - You may not want a subset of sensitive data can be queryable by all users.
 - You can use a database view to expose only non-sensitive data to a specific group of users.
- Provides extra security layer
 - Allows you to create the read-only view to expose read-only data to specific users.

Views: Advantages

- Enables computed columns.
 - A database table should not have calculated columns (3NF) however a database view should.
- Enables backward compatibility

Views: Disadvantages

- Performance: querying data from a database view can be slow especially if the view is created based on other views.
- Tables dependency: you create a view based on underlying tables of the database. Whenever you change the structure of these tables that view associated with, you have to change the view as well.

Creating Views

CREATE

[ALGORITHM = {MERGE | TEMPTABLE | UNDEFINED}]

VIEW [database_name].[view_name]

AS

[SELECT statement]

MERGE algorithm

- MySQL first combines the input query with the SELECT statement, which defines the view, into a single query.
- Executes the combined query to return the result set.
- The MERGE algorithm is not allowed if the SELECT statement contains aggregate functions such as MIN, MAX, SUM, COUNT, AVG or DISTINCT, GROUP BY, HAVING, LIMIT, UNION, UNION ALL, subquery.

TEMPTABLE algorithm

- MySQL first creates a temporary table based on the SELECT statement that defines the view
- Executes the input query against this temporary table.
- TEMPTABLE algorithm is less efficient than the MERGE algorithm.
- A view that uses TEMPTABLE algorithm is not updatable.

UNDEFINED algorithm

- The default algorithm when you create a view without specifying an explicit algorithm.
- The UNDEFINED algorithm lets MySQL make a choice of using MERGE or TEMPTABLE algorithm.
- MySQL prefers MERGE algorithm to TEMPTABLE algorithm because the MERGE algorithm is much more efficient.

CREATE VIEW EXAMPLE

```
CREATE VIEW SalePerOrder AS
```

```
SELECT
```

```
    orderNumber, SUM(quantityOrdered * priceEach) total
```

```
FROM
```

```
    orderDetails
```

```
GROUP by orderNumber
```

```
ORDER BY total DESC;
```


Variables

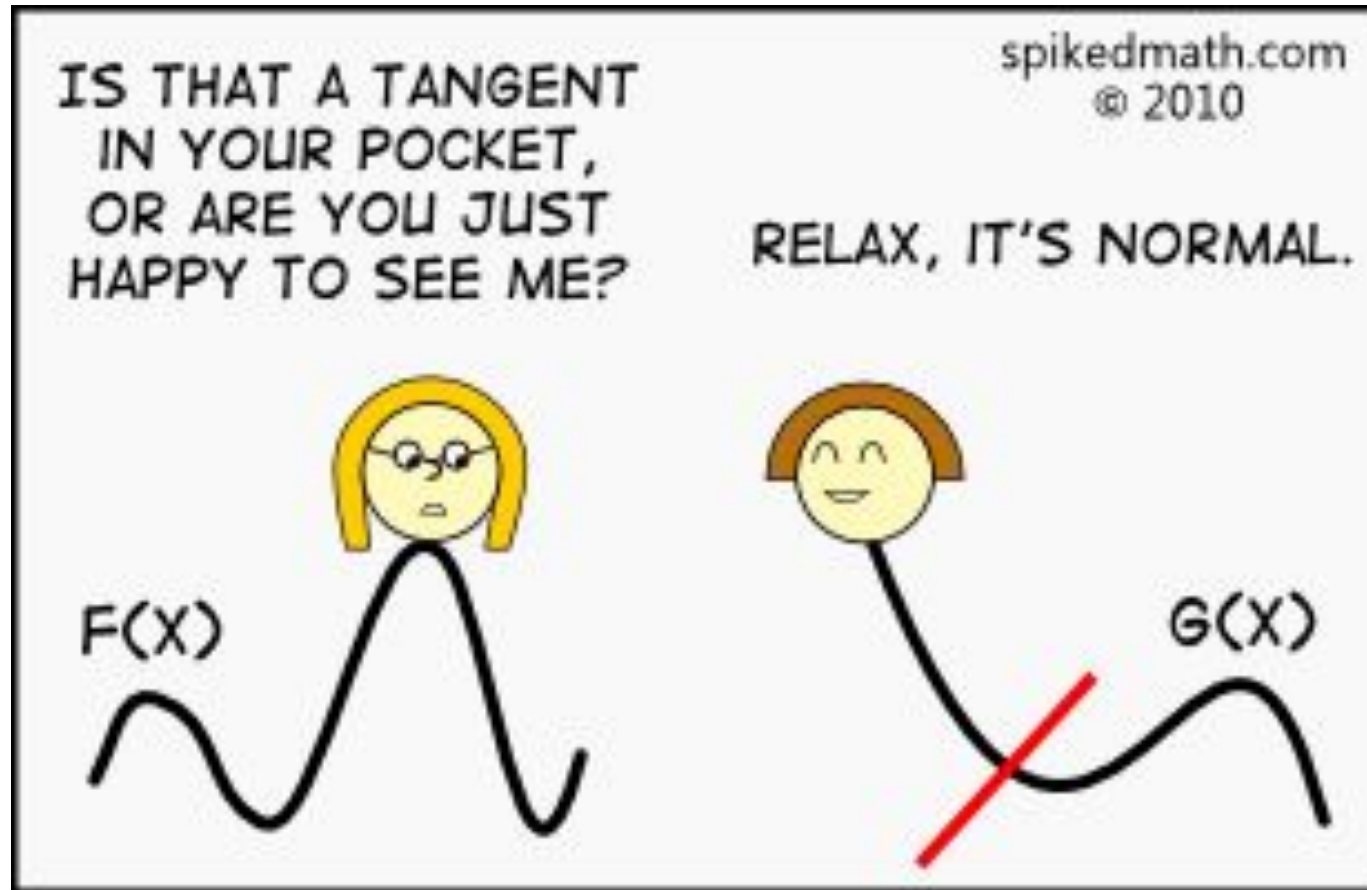
Views

Functions

Stored Procedures

Triggers

Functions



The idea of a function

A function receives input data, do some processing and gives an output



The idea of a function



Stored Functions

- **Stored Function** is a stored program that returns a **single value**.
- You can use stored functions to **encapsulate** common formulas or business rules that may be **reusable** among SQL statements or stored programs.
- You can use a stored function in SQL statements wherever an expression is used. This helps improve the readability and maintainability of the procedural code.

Stored Functions

- Functions cannot perform permanent environmental changes to SQL Server (i.e. no INSERT or UPDATE statements allowed).
- A Function can be used inline in SQL Statements if it returns a scalar value or can be joined upon if it returns a result set

Stored Functions



Function

```
SELECT * FROM orders WHERE YEAR(shippedDate) = 2003;
```

Create Function Syntax

CREATE FUNCTION function_name(param1,param2,...)

RETURNS datatype

[NOT] DETERMINISTIC

statements

Create Function Syntax

- Specify the name of the stored function after CREATE FUNCTION keywords.

CREATE FUNCTION *addTwoNumbers*

- List all parameters of the function.

CREATE FUNCTION *addTwoNumbers*(*num1 double, num2 double*)

Create Function Syntax

- Specify the data type of the return value in the RETURNS statement. It can be any valid MySQL data type.

```
CREATE FUNCTION addTwoNumbers(num1 double, num2 double)  
    RETURNS double
```

Create Function Syntax

- For the same input parameters, if the function returns the same result, it is considered deterministic and not deterministic otherwise.

```
CREATE FUNCTION addTwoNumbers(num1 double, num2 double)  
    RETURNS double  
    DETERMINISTIC
```

Create Function Syntax

- For the same input parameters, if the function returns the same result, it is considered deterministic and not deterministic otherwise.

```
CREATE FUNCTION addTwoNumbers(num1 double, num2 double)  
    RETURNS double  
    DETERMINISTIC
```

- You have to decide whether a stored function is deterministic or not. If you declare it incorrectly, the stored function may produced an unexpected result, or the available optimization is not used which degrade the performance.

Create Function Syntax

- Fifth, you write the code in the statements section. It can be a single statement or a compound statement. Inside the statements section, you have to specify at least one **RETURN** statement.
- The RETURN statement returns a value to the caller. Whenever the RETURN statement is reached, the stored function's execution is terminated immediately.

Create Function Syntax

DELIMITER \$\$

CREATE FUNCTION addTwoNumbers(num1 double, num2 double)

RETURNS double

DETERMINISTIC

BEGIN

DECLARE sumOfTwoNumbers double;

sumOfTwoNumbers = num1 * num2;

RETURN (sumOfTwoNumbers);

END

Create Function Syntax

DELIMITER \$\$

CREATE FUNCTION customerLevel(p_creditLimit double) RETURNS VARCHAR(10)

DETERMINISTIC

BEGIN

DECLARE lvl varchar(10);

IF p_creditLimit > 50000 THEN

SET lvl = 'PLATINUM';

ELSEIF (p_creditLimit <= 50000 AND p_creditLimit >= 10000) THEN

SET lvl = 'GOLD';

ELSEIF p_creditLimit < 10000 THEN

SET lvl = 'SILVER';

END IF;

RETURN (lvl);

END

Variables

Views

Functions

Stored Procedures

Definition of Stored Procedures

- Stored Procedure (SP) is a segment of declarative SQL statements stored inside the database catalog.
- SP can be invoked by triggers, other stored procedures or applications such as Java, C#, PHP, etc.
- SP that calls itself is known as a **recursive stored procedure**. MySQL does not support recursion very well.

SP Advantages

- SPs help increase the performance of the applications. Once created, stored procedures are compiled and stored in the database.
- MySQL SPs are compiled on demand.
 - After compiling a stored procedure, MySQL puts it to a cache.
 - MySQL maintains its own SP cache for every single connection.
 - If an application uses a stored procedure multiple times in a single connection, the compiled version is used, otherwise the stored procedure works like a query.

SP Advantages

- SPs helps reduce the traffic between application and database server
 - Instead of sending multiple lengthy SQL statements, the application has to send only name and parameters of the stored procedure.
- SPs are reusable and transparent to any applications.
 - Stored procedures expose the database interface to all applications so that developers don't have to develop functions that are already supported in stored procedures.

SP Advantages

- SPs are more secure than SQL statements or scripts.
 - Database administrator can grant appropriate permissions to applications that access stored procedures in the database without giving any permission on the underlying database tables.

SP Disadvantages

- If you use a lot of SPs, the memory usage of every connection that is using those stored procedures will increase substantially.
- If you overuse a large number of logical operations inside SPs, the CPU usage will also increase because database server is not well-designed for logical operations.

SP Disadvantages

- It is difficult to debug stored procedures. Only few database management systems allow you to debug stored procedures. Unfortunately, MySQL does not provide facilities for debugging stored procedures.

CREATE STORED PROCEDURE

```
CREATE PROCEDURE GetAllProducts()  
  BEGIN  
    SELECT * FROM products;  
  END;
```

CREATE STORED PROCEDURE

- **CREATE PROCEDURE** statement creates a new stored procedure.
- Specify the name of SP after the CREATE PROCEDURE statement. In this case, the name of the stored procedure is GetAllProducts.
- We put the parentheses after the name of the stored procedure.

CREATE STORED PROCEDURE

- The section between BEGIN and END is called the body of the stored procedure.
- You put the declarative SQL statements in the body to handle business logic.
- In this stored procedure, we use a simple SELECT statement to query data from the products table.

CALL STORED PROCEDURE

CALL STORED_PROCEDURE_NAME()

- You use the CALL statement to call a stored procedure e.g., to call the *GetAllProducts* stored procedure, you use the following statement:

CALL GetAllProducts();

DECLARE SP VARIABLES

To declare a variable inside a stored procedure, you use the **DECLARE** statement as follows:

```
DECLARE variable_name datatype(size) DEFAULT default_value;
```

DECLARE SP VARIABLES

DECLARE variable_name datatype(size) DEFAULT default_value;

- Specify the variable name after the DECLARE keyword.
 - The variable name must follow the naming rules of MySQL table column names.
- Specify the data type of the variable and its size.
 - A variable can have any MySQL data types such as INT, VARCHAR, DATETIME, etc.

DECLARE SP VARIABLES

- When you declare a variable, its initial value is NULL.
 - You can assign the variable a default value by using DEFAULT keyword.

```
DECLARE total_sale INT DEFAULT 0
```

DECLARE SP VARIABLES

- Once you declared a variable, you can start using it. To assign a variable another value, you use the SET statement, for example:

```
DECLARE total_count INT DEFAULT 0  
SET total_count = 10;
```

SP PARAMETERS

- Almost stored procedures that you develop require parameters.
- The parameters make the stored procedure more flexible and useful.
- In MySQL, a parameter has one of three modes
 - IN
 - OUT
 - INOUT

SP PARAMETERS - IN

- IN is the default mode.
- When you define an IN parameter in a SP, the calling program has to pass an argument to the SP.
- The value of an IN parameter is protected.
 - It means that even the value of the IN parameter is changed inside the SP, its original value is retained after the SP ends.
 - The SP only works on the copy of the IN parameter.

SP PARAMETERS - OUT

- OUT – the value of an OUT parameter can be changed inside the SP and its new value is passed back to the calling program.
- It is the equivalent of the RETURN statement in a function

SP PARAMETERS - INOUT

- INOUT – an INOUT parameter is the combination of IN parameter and OUT parameter.
- The calling program may pass the argument, and the stored procedure can modify the INOUT parameter and pass the new value back to the calling program.

IN PARAMETER EXAMPLE

```
CREATE PROCEDURE GetOfficeByCountry  
    (IN countryName VARCHAR(255))  
  
BEGIN  
    SELECT * FROM offices  
    WHERE country = countryName;  
END;
```

OUT PARAMETER EXAMPLE

```
CREATE PROCEDURE CountOrderByStatus(  
    IN orderStatus VARCHAR(25),  
    OUT total INT)  
BEGIN  
    SELECT count(orderNumber)  
    INTO total  
    FROM orders  
    WHERE status = orderStatus;  
END;
```

OUT PARAMETER EXAMPLE

To get the number of shipped orders, we call the CountOrderByStatus stored procedure and pass the order status as Shipped, and also pass an argument (@total) to get the return value.

```
CALL CountOrderByStatus('Shipped',@total);  
SELECT @total;
```

INOUT PARAMETER EXAMPLE

```
CREATE PROCEDURE set_counter  
    (INOUT count INT(4),  
    IN inc INT(4))  
BEGIN  
    SET count = count + inc;  
END;
```

INOUT PARAMETER EXAMPLE

- The *set_counter* SP accepts:
 - one INOUT parameter (count)
 - one IN parameter (inc).
- Inside the stored procedure, we increase the counter (count) by the value of the inc parameter.

```
SET @counter = 1;  
CALL set_counter(@counter,1); -- 2  
CALL set_counter(@counter,1); -- 3  
CALL set_counter(@counter,5); -- 8  
SELECT @counter; -- 8
```