Stored Routines

365 DataScience

Introduction to Stored Routines

365 DataScience

Introduction to Stored Routines

routine *(in a context other than computer science)*

a usual, fixed action, or series of actions, repeated periodically

Introduction to Stored Routines

```
SELECT
    emp_no, MAX(from_date) AS from_date, MAX(to_date)
FROM
    dept_emp
GROUP BY emp_no;
```

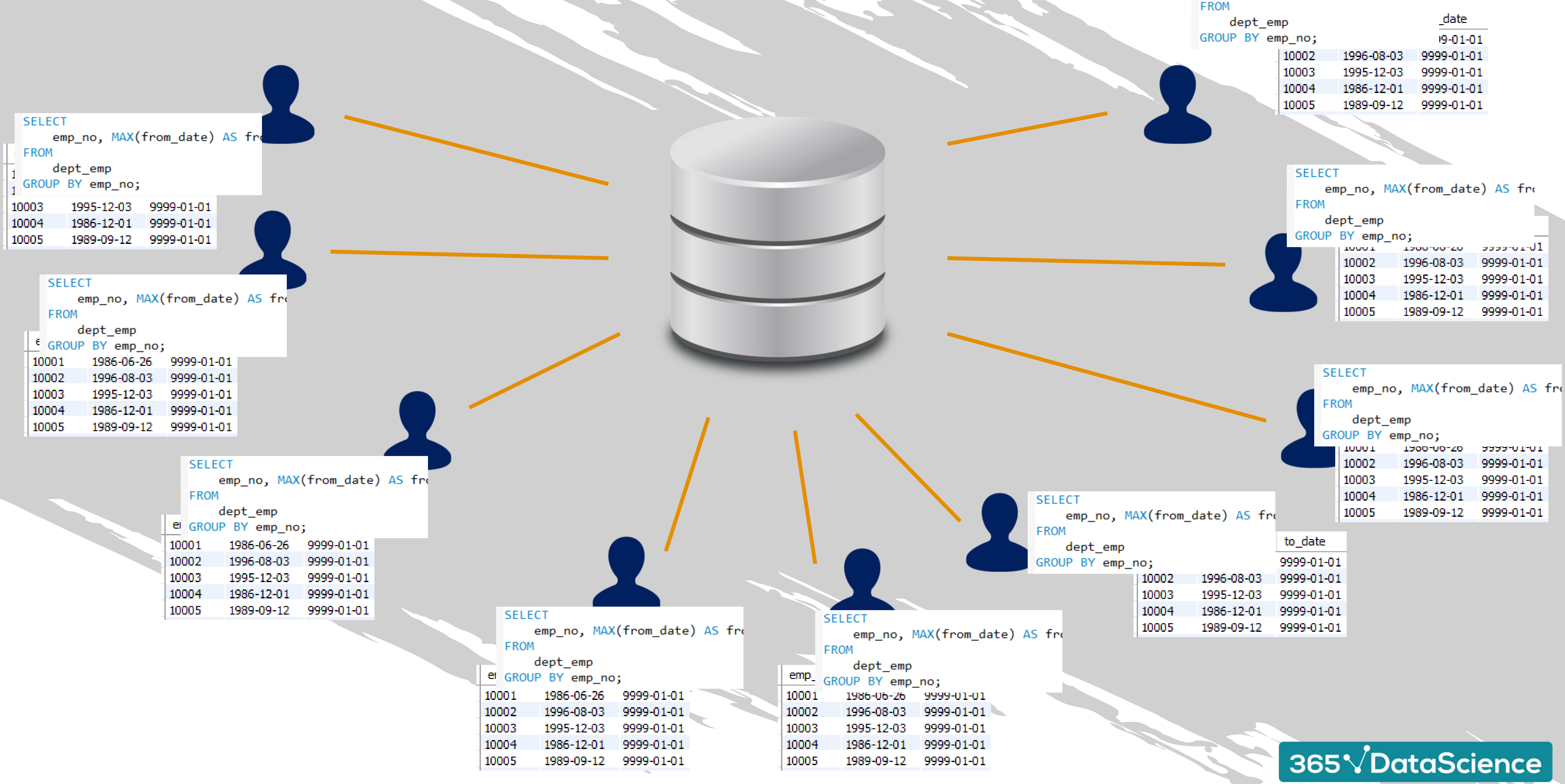
query

emp_no	from_date	to_date
10001	1986-06-26	9999-01-01
10002	1996-08-03	9999-01-01
10003	1995-12-03	9999-01-01
10004	1986-12-01	9
10005	1989-09-12	9

output

365 DataScience

Introduction to Stored Routines



365√DataScience

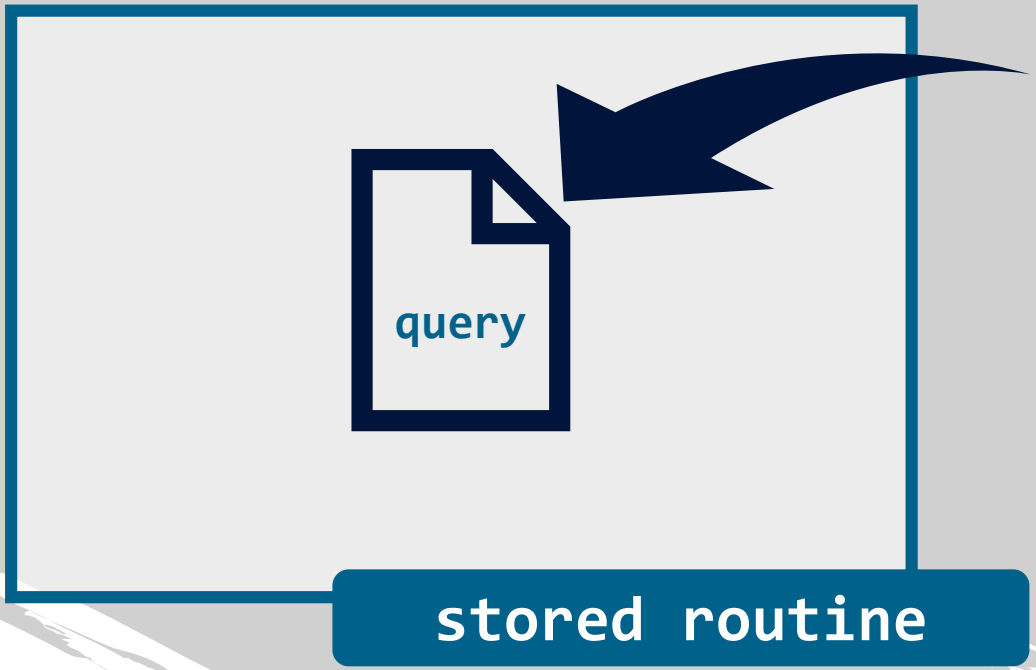
Introduction to Stored Routines

stored routine

an SQL statement, or a set of SQL statements, that can be stored on the database server

- whenever a user needs to run the query in question, they can *call*, *reference*, or *invoke* the routine

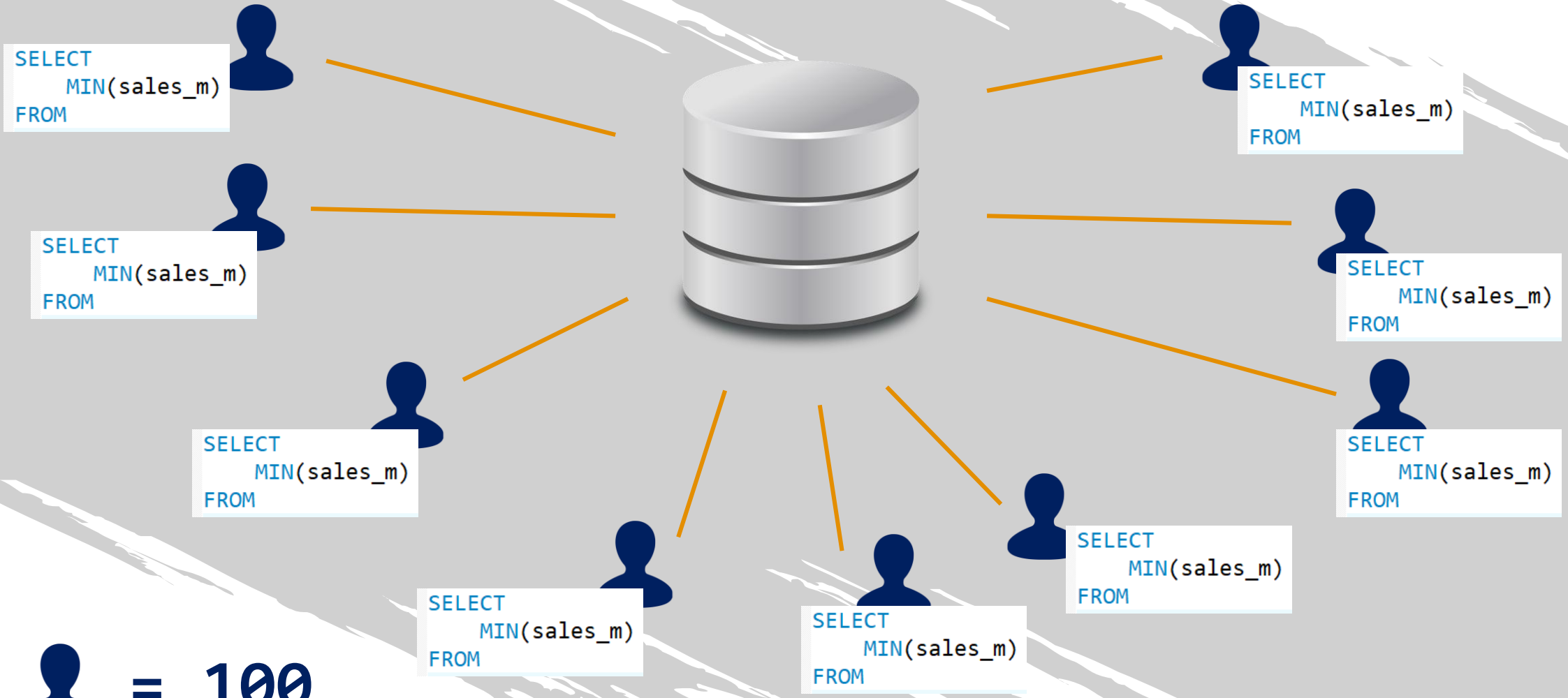
Introduction to Stored Routines



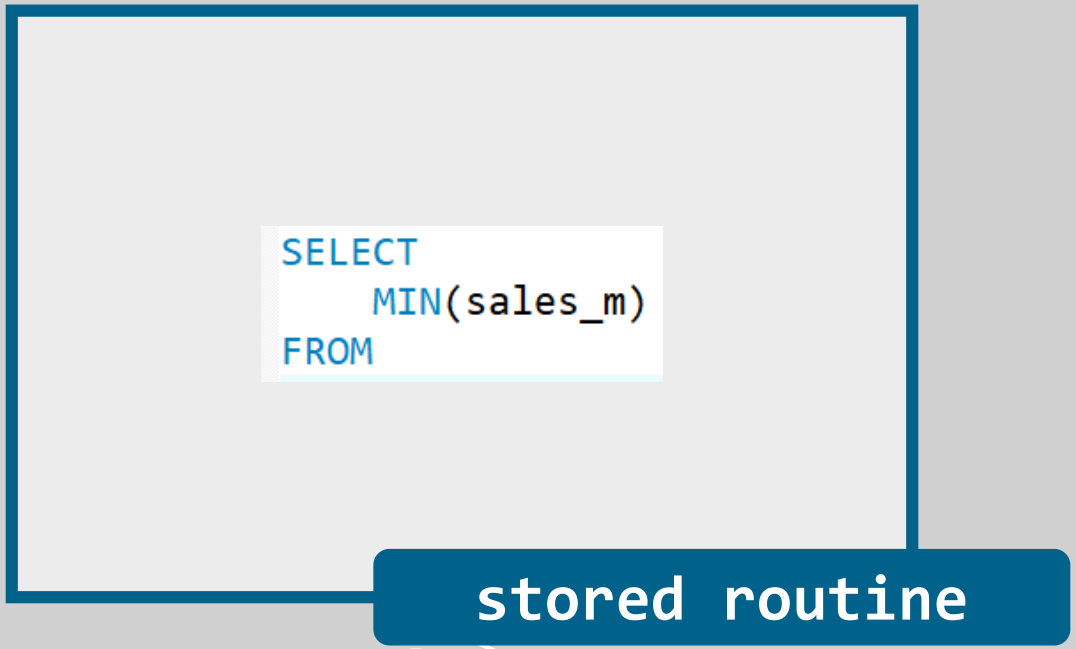
algorithm

1) *checks all monthly sales generated throughout a calendar year*
2) *returns the lowest of these values*

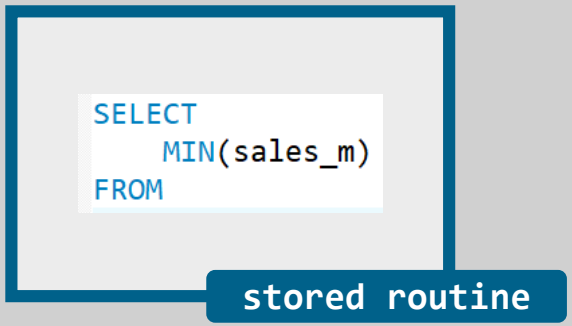
Introduction to Stored Routines



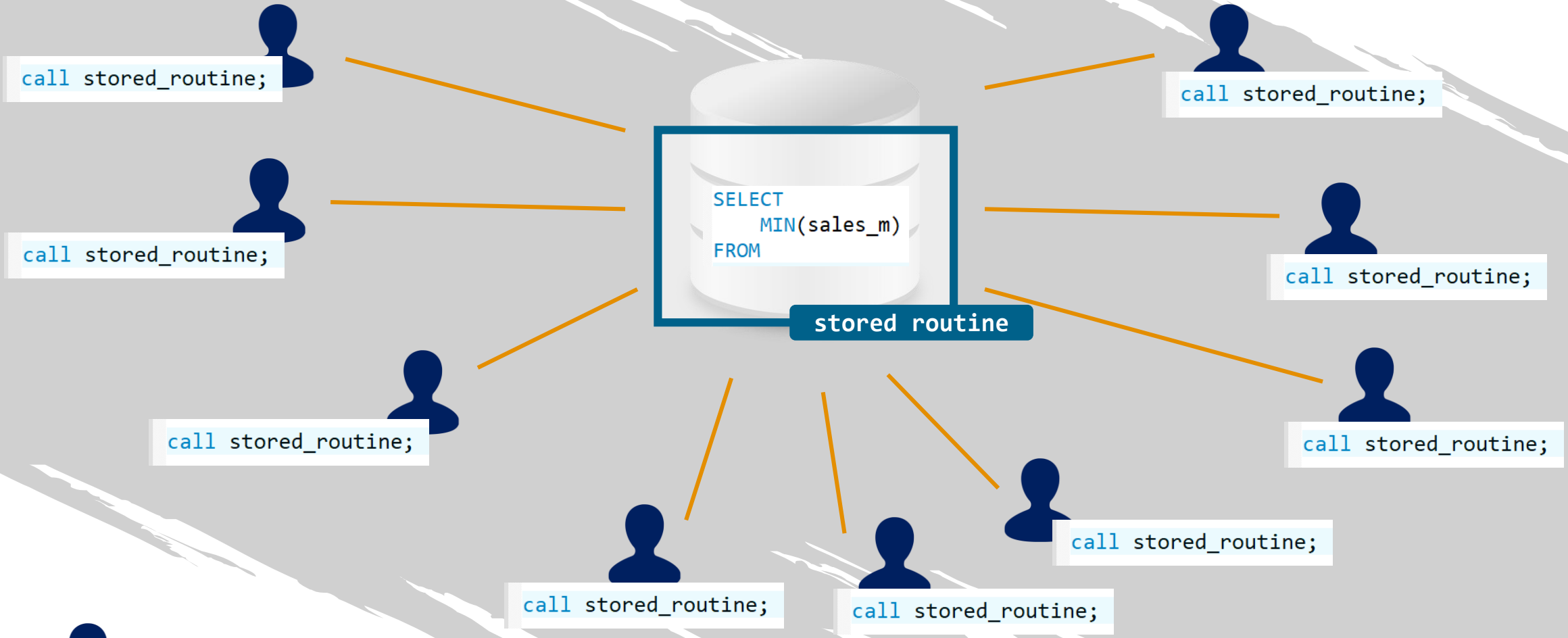
Introduction to Stored Routines



Introduction to Stored Routines



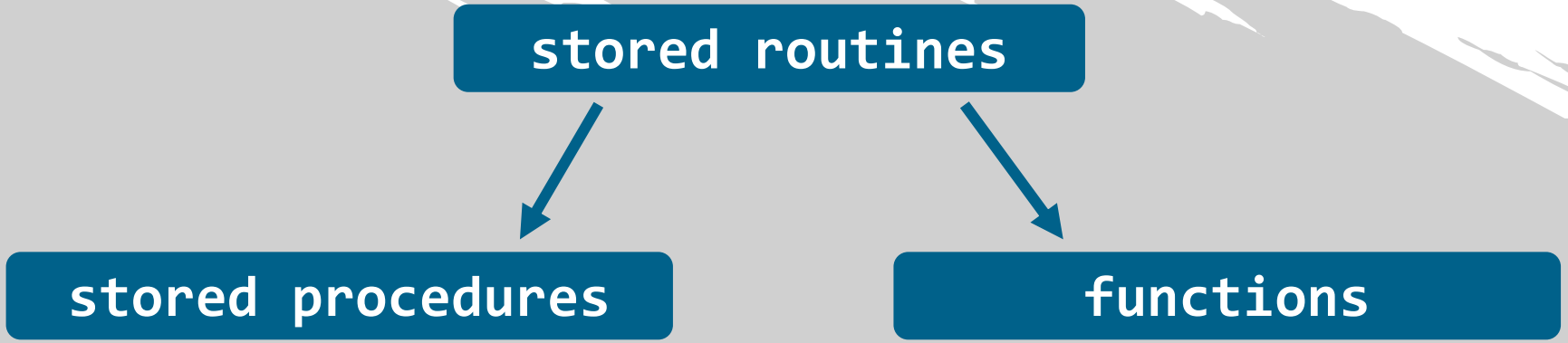
Introduction to Stored Routines



= 100 - this routine can bring the desired result multiple times

365 DataScience

Introduction to Stored Routines



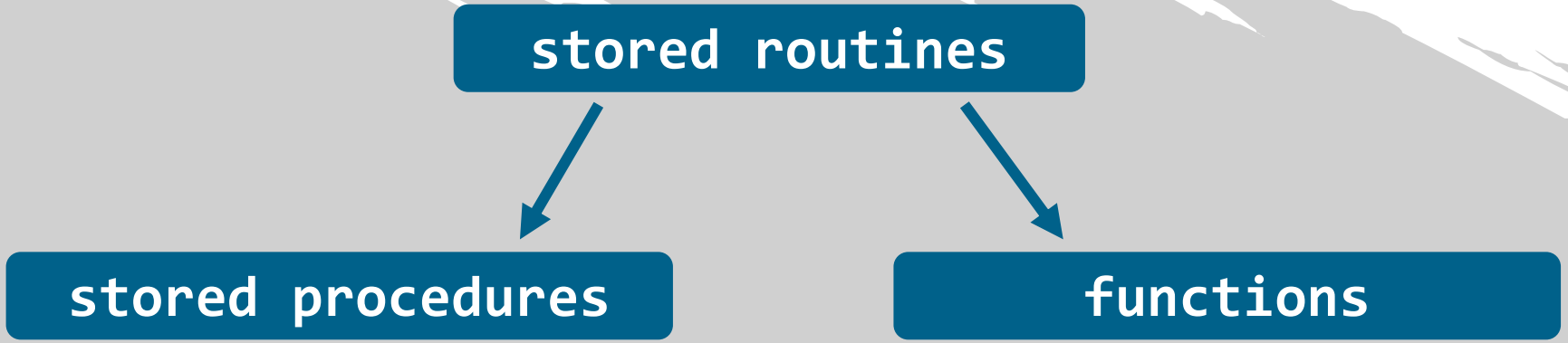
= procedures

= user-defined functions

≠ *built-in functions*

(aggregate functions, datetime functions)

Introduction to Stored Routines




The MySQL Syntax for Stored Procedures
365 DataScience

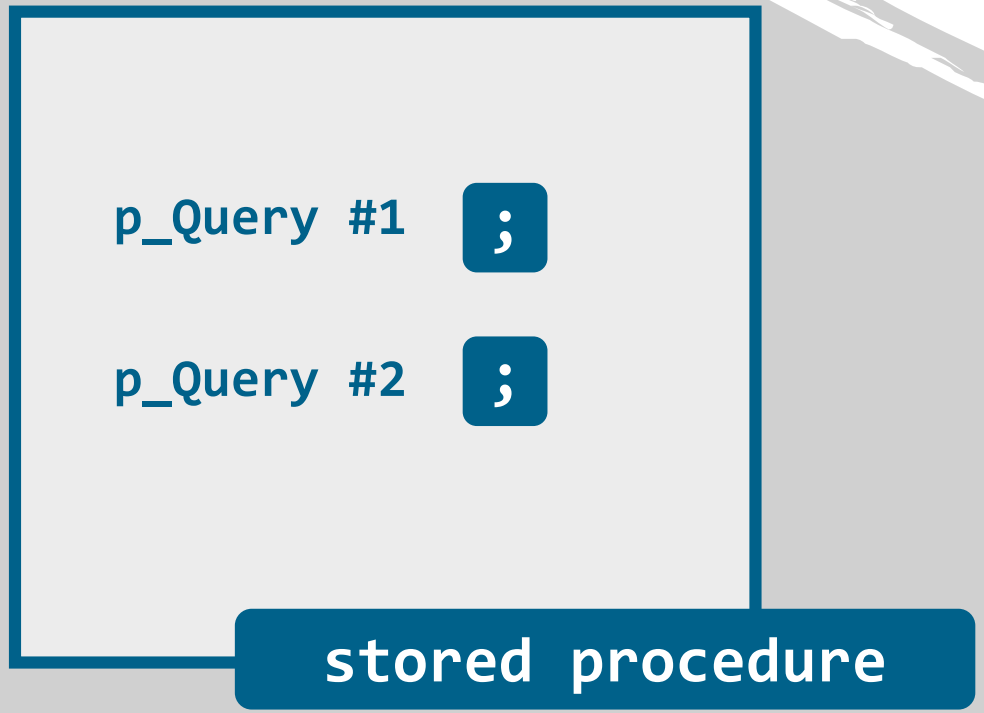
The MySQL Syntax for Stored Procedures

- semi-colons ;
  - they function as a statement terminator
  - technically, they can also be called delimiters
  - by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

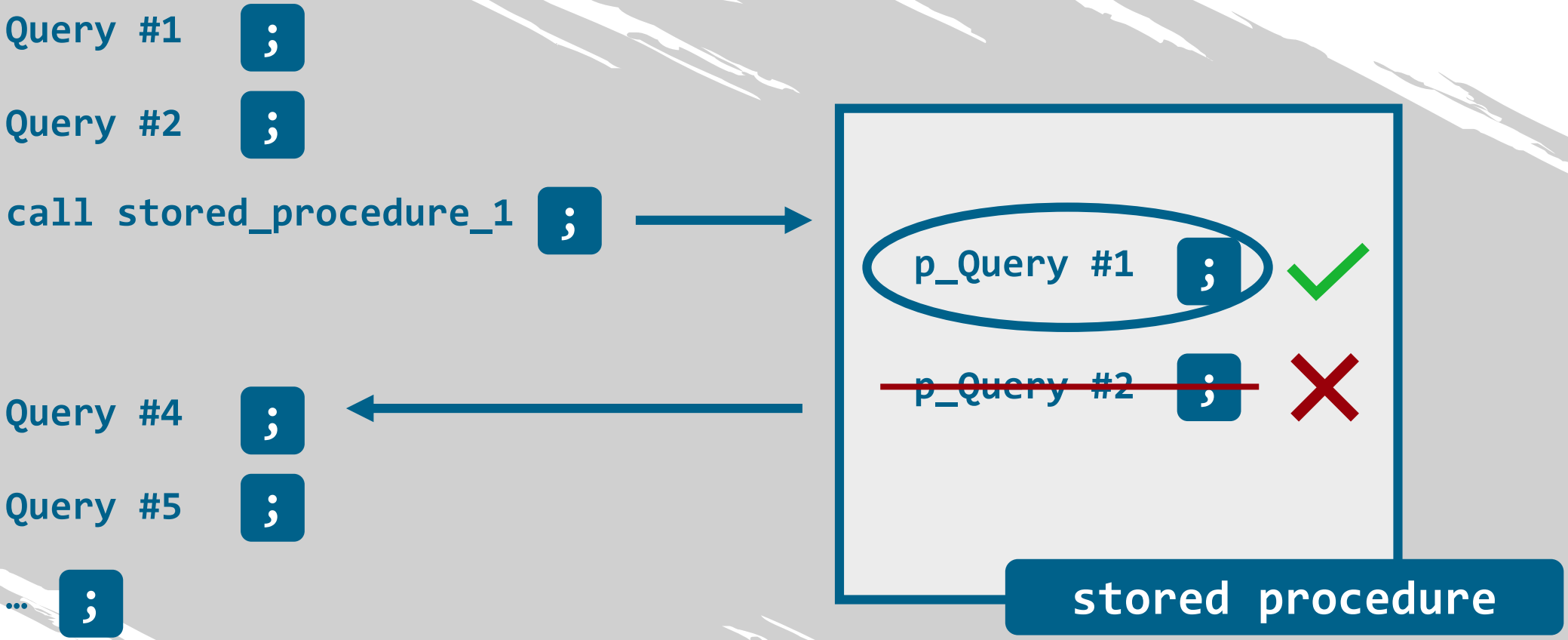

```SQL
DELIMITER $$
```

The MySQL Syntax for Stored Procedures

```
Query #1 ;
Query #2 ;
call stored_procedure_1 ;
```



The MySQL Syntax for Stored Procedures

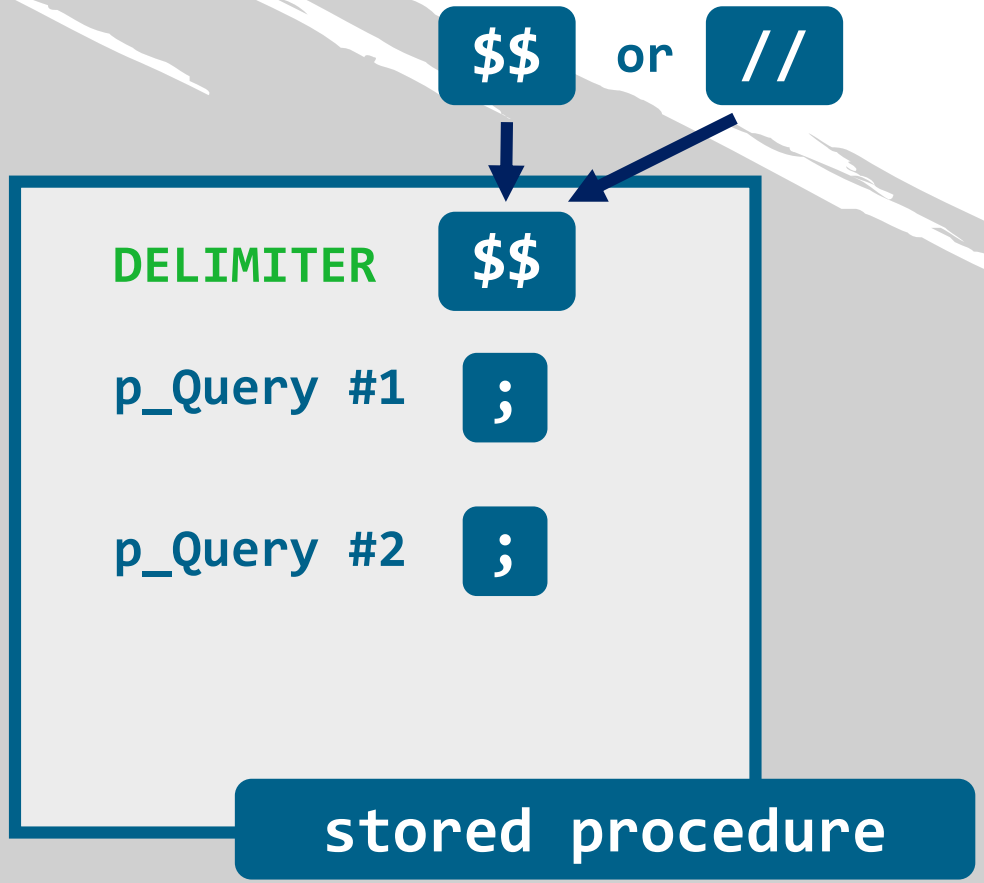


The MySQL Syntax for Stored Procedures

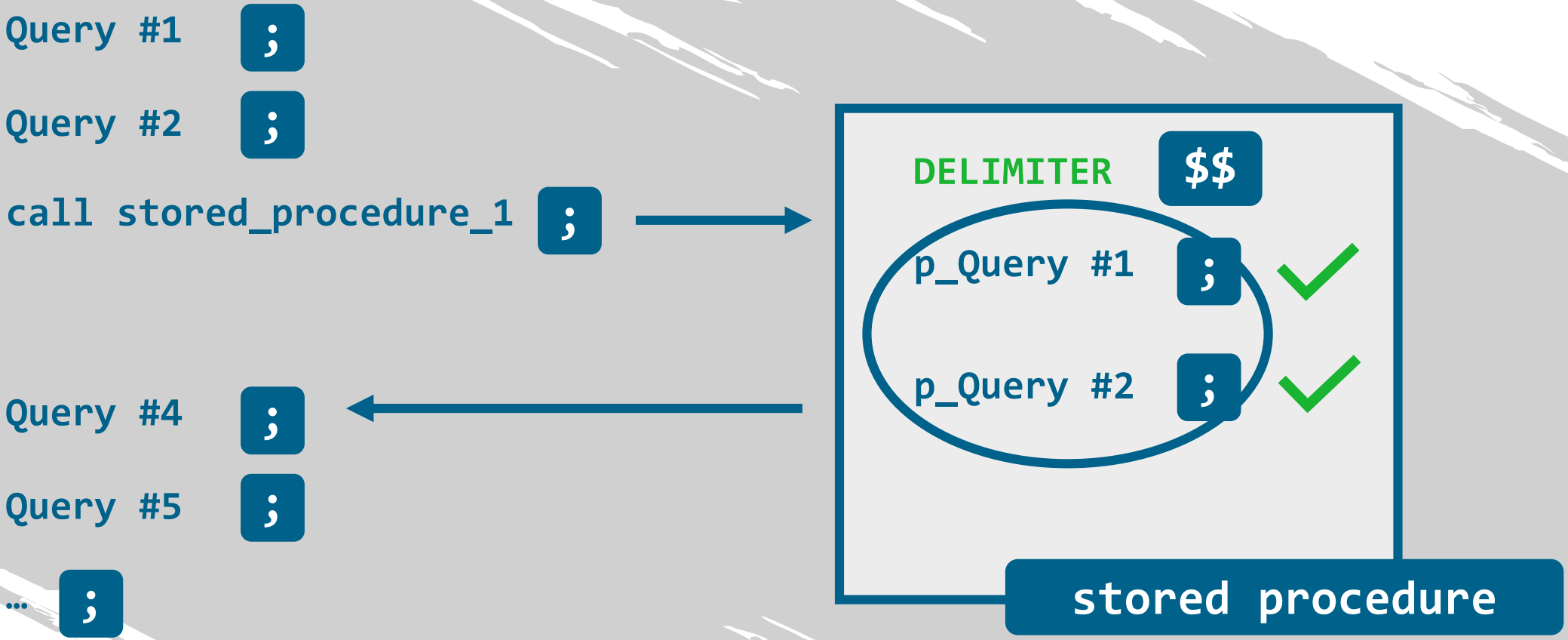
Query #1 ;

Query #2 ;

call stored_procedure_1 ;



The MySQL Syntax for Stored Procedures



The MySQL Syntax for Stored Procedures

SQL

```
DELIMITER $$

CREATE PROCEDURE procedure_name()
```

The MySQL Syntax for Stored Procedures


```
DELIMITER $$

CREATE PROCEDURE procedure_name(param_1, param_2)
```

Parameters represent certain values that the procedure will use to complete the calculation it is supposed to execute

The MySQL Syntax for Stored Procedures


```
DELIMITER $$

CREATE PROCEDURE procedure_name()
```

A procedure can be created without parameters too!

Nevertheless, the parentheses must always be attached to its name

The MySQL Syntax for Stored Procedures


```
DELIMITER $$
CREATE PROCEDURE procedure_name()
BEGIN
    SELECT * FROM employees
    LIMIT 1000;
END$$
```

The MySQL Syntax for Stored Procedures


```
DELIMITER $$
CREATE PROCEDURE procedure_name()
BEGIN
    SELECT * FROM employees
    LIMIT 1000;
END$$
```

body of the procedure

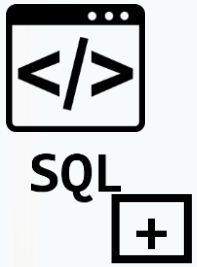
The MySQL Syntax for Stored Procedures


```
DELIMITER $$
CREATE PROCEDURE procedure_name()
BEGIN
    SELECT * FROM employees
    LIMIT 1000;
END$$
```

The MySQL Syntax for Stored Procedures


```
DELIMITER $$
CREATE PROCEDURE procedure_name()
BEGIN
    SELECT * FROM employees
    LIMIT 1000;
END$$
```

The MySQL Syntax for Stored Procedures

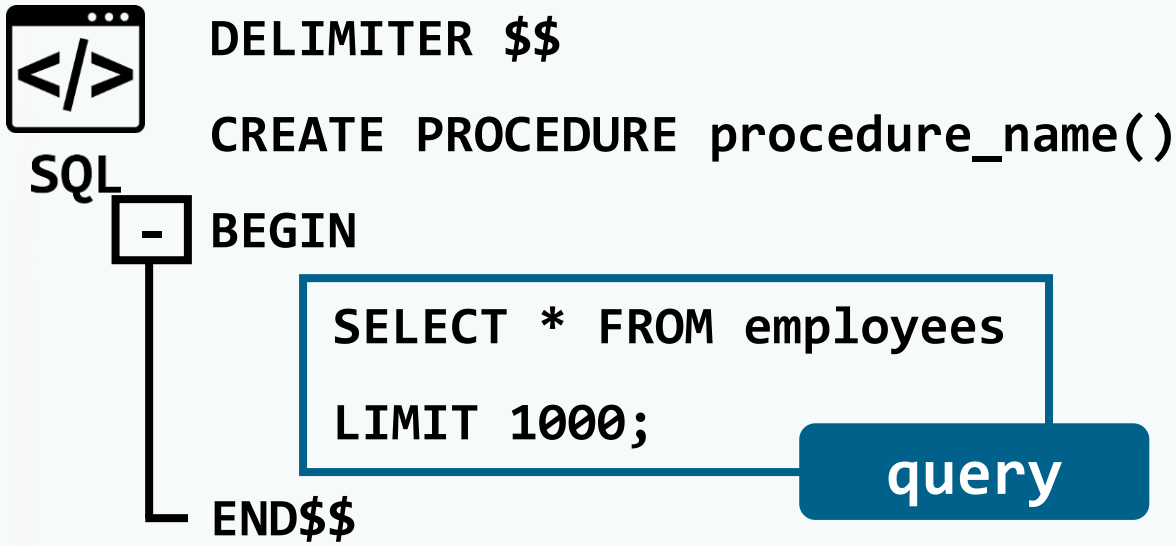



```
DELIMITER $$

CREATE PROCEDURE procedure_name()

BEGIN
```

The MySQL Syntax for Stored Procedures


```
DELIMITER $$
CREATE PROCEDURE procedure_name()
BEGIN
    SELECT * FROM employees
    LIMIT 1000;
END$$
```

The MySQL Syntax for Stored Procedures

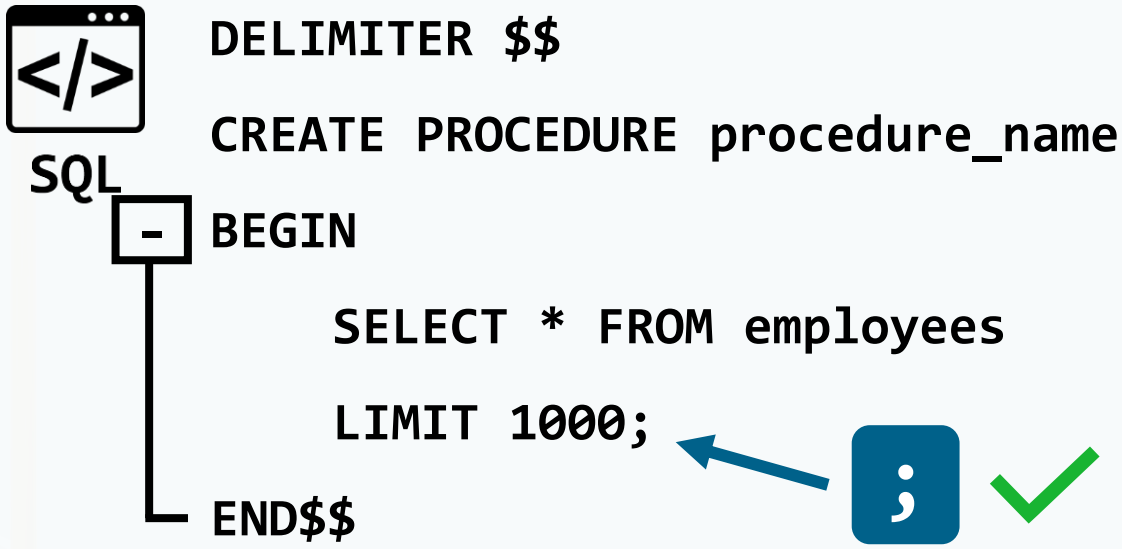


The MySQL Syntax for Stored Procedures


```
DELIMITER $$
CREATE PROCEDURE procedure_name()
BEGIN
    SELECT * FROM employees
    LIMIT 1000;
END$$
```

;

The MySQL Syntax for Stored Procedures



The MySQL Syntax for Stored Procedures


```
DELIMITER $$
CREATE PROCEDURE procedure_name()
BEGIN
    SELECT * FROM employees
    LIMIT 1000$$
END$$
```

The MySQL Syntax for Stored Procedures


```
DELIMITER $$
CREATE PROCEDURE procedure_name()
BEGIN
    SELECT * FROM employees
    LIMIT 1000$$
END$$
```

The MySQL Syntax for Stored Procedures


```
DELIMITER $$
CREATE PROCEDURE procedure_name()
BEGIN
    SELECT * FROM employees
    LIMIT 1000;
END$$
```

The MySQL Syntax for Stored Procedures


```
DELIMITER $$
CREATE PROCEDURE procedure_name()
BEGIN
    SELECT * FROM employees
    LIMIT 1000;
END$$
DELIMITER ;
```

The MySQL Syntax for Stored Procedures


```
DELIMITER $$
CREATE PROCEDURE procedure_name()
BEGIN
    SELECT * FROM employees
    LIMIT 1000;
END$$
DELIMITER ;
```

From this moment on, $$ will not act as a delimiter

Stored Procedures with an Input Parameter

365 DataScience

Stored Procedures with an Input Parameter

- a stored routine can perform a calculation that transforms an *input* value in an *output* value

Stored Procedures with an Input Parameter

- a stored routine can perform a calculation that transforms an *input* value in an *output* value
- stored procedures can take an *input* value and then use it in the query, or queries, written in the body of the procedure

Stored Procedures with an Input Parameter

- a stored routine can perform a calculation that transforms an *input* value in an *output* value
- stored procedures can take an *input* value and then use it in the query, or queries, written in the body of the procedure

- this value is represented by the IN parameter

Stored Procedures with an Input Parameter

- a stored routine can perform a calculation that transforms an *input* value in an *output* value
- stored procedures can take an *input* value and then use it in the query, or queries, written in the body of the procedure

- this value is represented by the IN parameter

Stored Procedures with an Input Parameter

- a stored routine can perform a calculation that transforms an *input* value in an *output* value
- stored procedures can take an *input* value and then use it in the query, or queries, written in the body of the procedure

- this value is represented by the IN parameter

Stored Procedures with an Input Parameter

- a stored routine can perform a calculation that transforms an *input* value in an *output* value
- stored procedures can take an *input* value and then use it in the query, or queries, written in the body of the procedure

- this value is represented by the IN parameter

Stored Procedures with an Input Parameter


```
DELIMITER $$
CREATE PROCEDURE procedure_name(in parameter)
BEGIN
    SELECT * FROM employees
    LIMIT 1000;
END$$
DELIMITER ;
```

Stored Procedures with an Input Parameter

- a stored routine can perform a calculation that transforms an *input* value in an *output* value
- stored procedures can take an *input* value and then use it in the query, or queries, written in the body of the procedure

- this value is represented by the IN parameter

Stored Procedures with an Input Parameter

- a stored routine can perform a calculation that transforms an *input* value in an *output* value
- stored procedures can take an *input* value and then use it in the query, or queries, written in the body of the procedure

- this value is represented by the IN parameter

- after that calculation is ready, a result will be returned

Stored Procedures with an Output Parameter

365 DataScience

Stored Procedures with an Output Parameter


```
DELIMITER $$
CREATE PROCEDURE procedure_name(in parameter, out parameter)
BEGIN
    SELECT * FROM employees
    LIMIT 1000;
END$$
DELIMITER ;
```

Stored Procedures with an Output Parameter


```
DELIMITER $$
CREATE PROCEDURE procedure_name(in parameter, out_parameter)
BEGIN
    SELECT * FROM employees
    LIMIT 1000;
END$$
DELIMITER ;
```

Stored Procedures with an Output Parameter


```
DELIMITER $$
CREATE PROCEDURE procedure_name(in parameter, out parameter)
BEGIN
    SELECT * FROM employees
    LIMIT 1000;
END$$
DELIMITER ;
```

Stored Procedures with an Output Parameter

SQL

```
DELIMITER $$
CREATE PROCEDURE procedure_name(in parameter, out parameter)
BEGIN
    SELECT * FROM employees
    LIMIT 1000;
END$$
DELIMITER ;
```

it will represent the variable containing the output value of the operation executed by the query of the stored procedure

Stored Procedures with an Output Parameter

- every time you create a procedure containing both an IN and an OUT parameter, remember that you must use the SELECT-INTO structure in the query of this object's body!

Variables

365 DataScience

Variables

- when you are *defining* a program, such as a stored procedure for instance, you can say you are using *'parameters'*

Variables

- when you are *defining* a program, such as a stored procedure for instance, you can say you are using *'parameters'*

  - *'parameters'* are a more abstract term

Variables

when you are *defining* a program, such as a stored procedure for instance, you can say you are using *'parameters'*

- *'parameters'* are a more abstract term

SQL

```
DELIMITER $$
CREATE PROCEDURE procedure_name (in                , out                )
```

Variables

- when you are *defining* a program, such as a stored procedure for instance, you can say you are using *'parameters'*
  - *'parameters'* are a more abstract term

SQL

```
DELIMITER $$
CREATE PROCEDURE procedure_name (in parameter, out                )
```

Variables

- when you are *defining* a program, such as a stored procedure for instance, you can say you are using *'parameters'*
  - *'parameters'* are a more abstract term


```sql
DELIMITER $$
CREATE PROCEDURE procedure_name (in parameter, out parameter)
```

Variables

- once the structure has been solidified, then it will be applied to the database. The input value you insert is typically referred to as the *‘argument’*, while the obtained output value is stored in a *‘variable’*

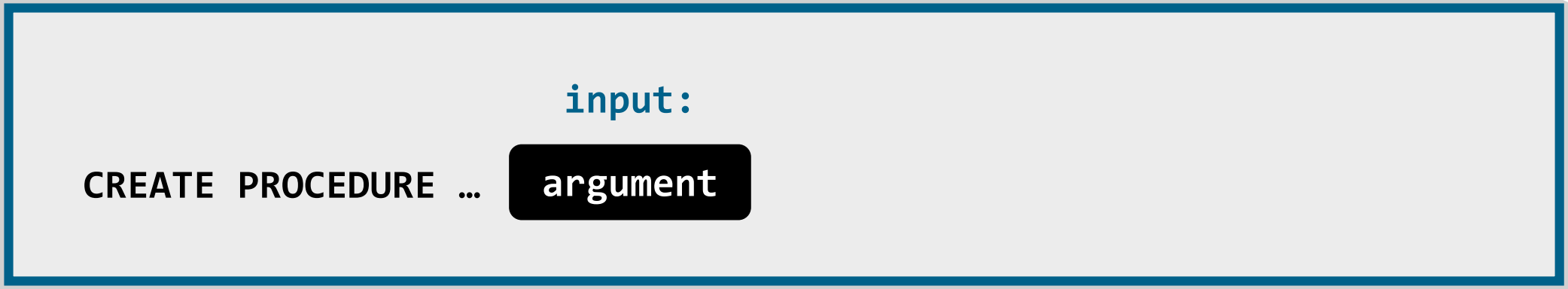
Variables

once the structure has been solidified, then it will be applied to the database. The input value you insert is typically referred to as the *'argument'*, while the obtained output value is stored in a *'variable'*

```
CREATE PROCEDURE …
```

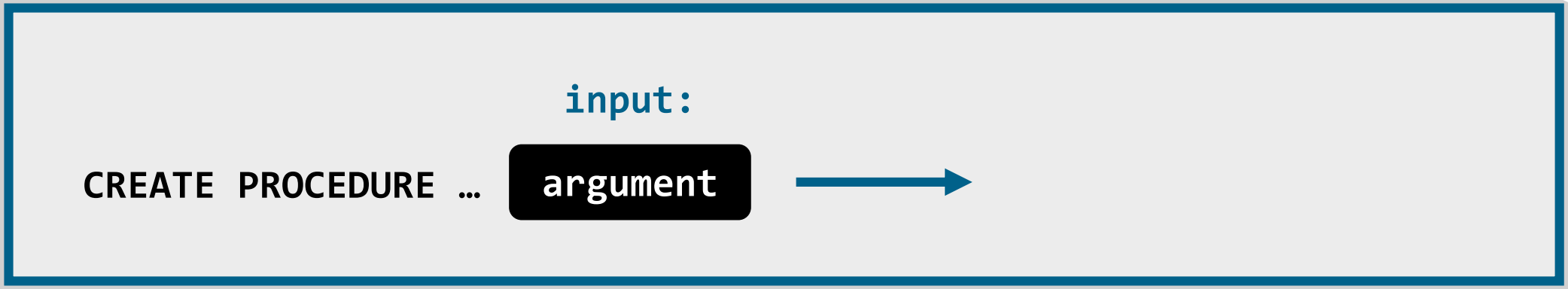
Variables

once the structure has been solidified, then it will be applied to the database. The input value you insert is typically referred to as the *'argument'*, while the obtained output value is stored in a *'variable'*



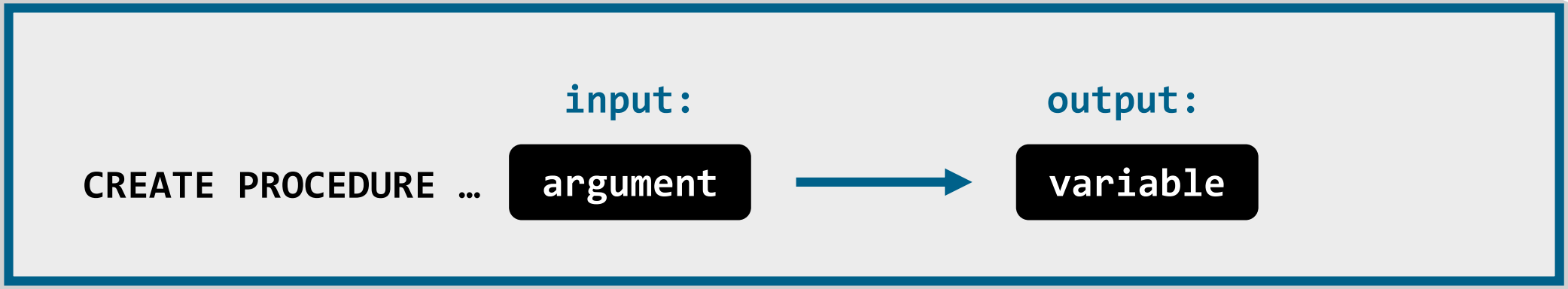
Variables

once the structure has been solidified, then it will be applied to the database. The input value you insert is typically referred to as the *'argument'*, while the obtained output value is stored in a *'variable'*



Variables

once the structure has been solidified, then it will be applied to the database. The input value you insert is typically referred to as the *'argument'*, while the obtained output value is stored in a *'variable'*



Variables

SQL

```
DELIMITER $$
CREATE PROCEDURE procedure_name (in parameter, out parameter)
```

≠

Variables

- IN-OUT parameters

```
CREATE PROCEDURE …
```

Variables

- IN-OUT parameters

```
CREATE PROCEDURE …
```

input:

IN parameter

Variables

- IN-OUT parameters

```
CREATE PROCEDURE …
```

input:

IN parameter

Variables

- IN-OUT parameters

```
CREATE PROCEDURE …
```

input:

OUT parameter

Variables

- IN-OUT parameters

input = output

```
CREATE PROCEDURE …
```

OUT parameter


User-Defined Functions in MySQL
365 DataScience

User-Defined Functions in MySQL


```sql
DELIMITER $$
CREATE FUNCTION function_name(parameter data_type) RETURNS data_type
DECLARE variable_name data_type
BEGIN
    SELECT …
RETURN variable_name
END$$
DELIMITER ;
```

User-Defined Functions in MySQL


```sql
DELIMITER $$
CREATE FUNCTION function_name(parameter data_type) RETURNS data_type
DECLARE variable_name data_type
BEGIN
    SELECT …
RETURN variable_name
END$$
DELIMITER ;
```

here you have no OUT parameters to define between the parentheses after the object's name

User-Defined Functions in MySQL


```sql
DELIMITER $$
CREATE FUNCTION function_name(parameter data_type) RETURNS data_type
DECLARE variable_name data_type
BEGIN
    SELECT …
RETURN variable_name
END$$
DELIMITER ;
```

here you have no OUT parameters to define between the parentheses after the object's name

all parameters are IN, and since this is well known, you need not explicitly indicate it with the word, 'IN'

User-Defined Functions in MySQL


```sql
DELIMITER $$
CREATE FUNCTION function_name(parameter data_type) RETURNS data_type
DECLARE variable_name data_type
BEGIN
    SELECT …
RETURN variable_name
END$$
DELIMITER ;
```

although there are no OUT parameters, there is a 'return value'

User-Defined Functions in MySQL


```sql
DELIMITER $$
CREATE FUNCTION function_name(parameter data_type) RETURNS data_type
DECLARE variable_name data_type
BEGIN
    SELECT …
RETURN variable_name
END$$
DELIMITER ;
```

although there are no OUT parameters, there is a 'return value'

it is obtained after running the query contained in the body of the function

User-Defined Functions in MySQL


```sql
DELIMITER $$
CREATE FUNCTION function_name(parameter data_type) RETURNS data_type
DECLARE variable_name data_type
BEGIN
    SELECT …
RETURN variable_name
END$$
DELIMITER ;
```

it can be of any data type

365 DataScience

User-Defined Functions in MySQL

- we cannot *call* a function!

User-Defined Functions in MySQL

- we cannot *call* a function!
- we can *select* it, indicating an input value within parentheses

User-Defined Functions in MySQL

- we cannot *call* a function!
- we can *select* it, indicating an input value within parentheses


```SQL
SELECT function_name(input_value);
```

Stored Routines - Conclusion

365 DataScience

Stored Routines - Conclusion

TECHNICAL DIFFERENCES

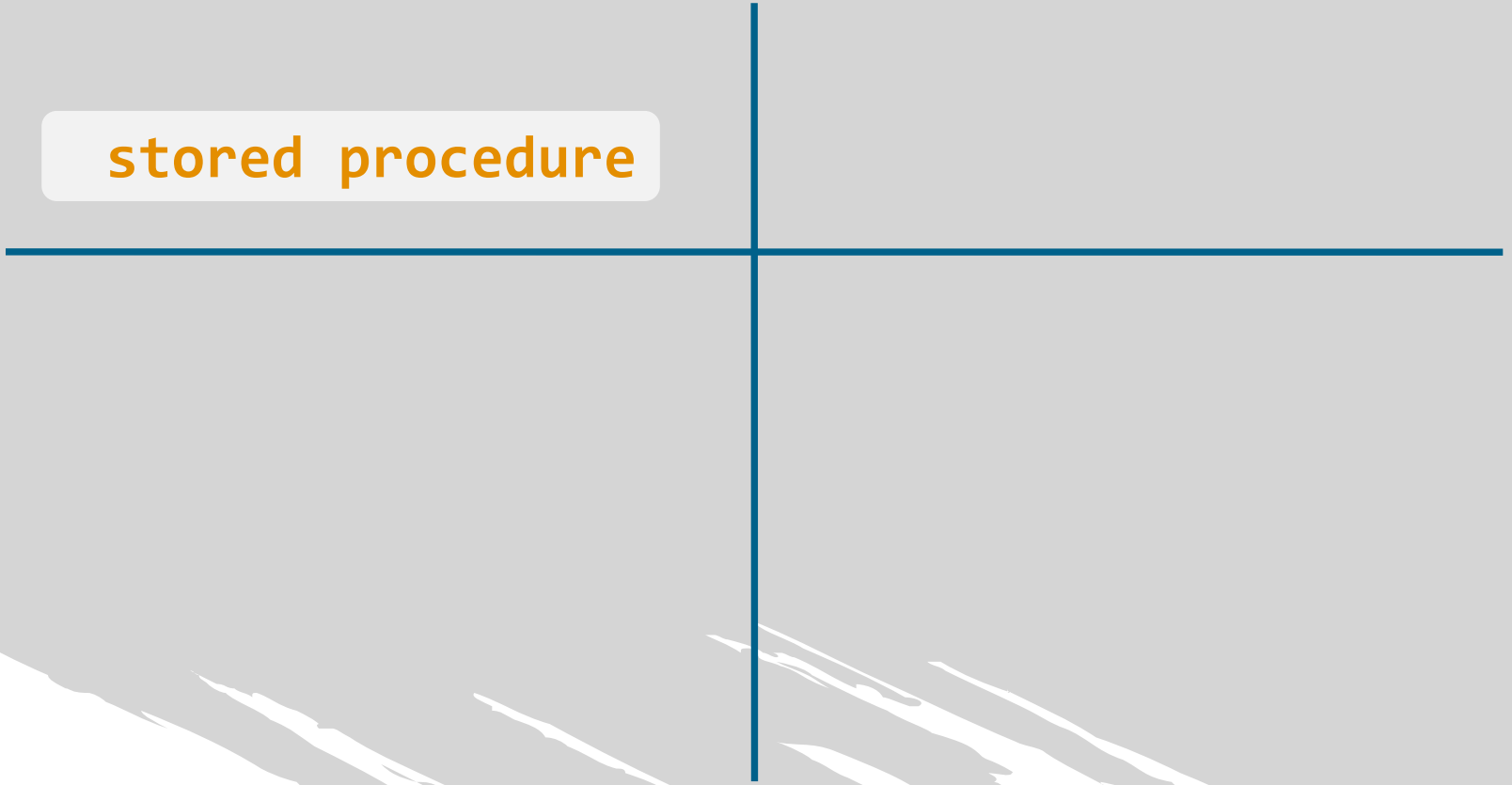
Stored Routines - Conclusion

TECHNICAL DIFFERENCES

`stored procedure`

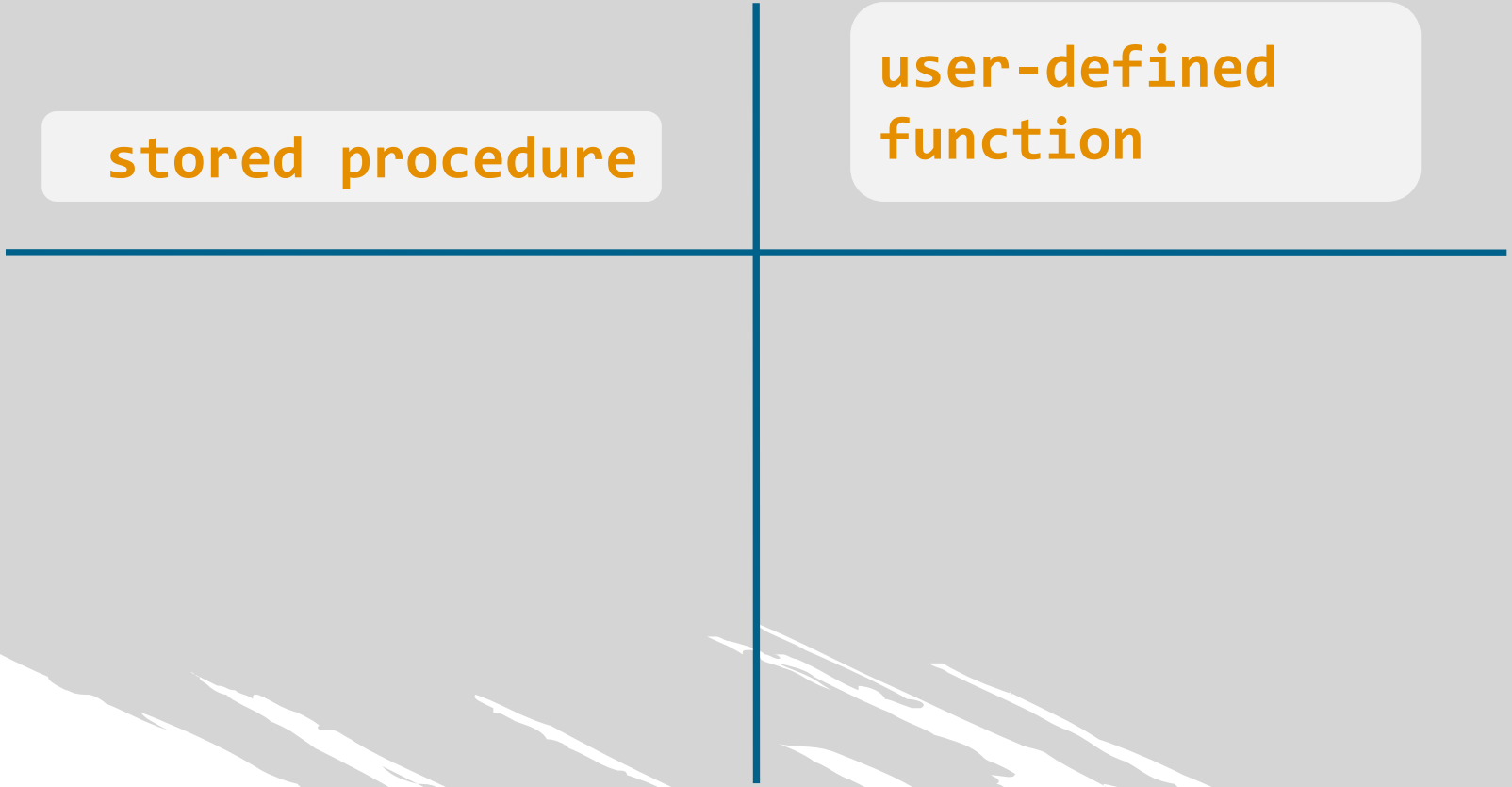
Stored Routines - Conclusion

TECHNICAL DIFFERENCES



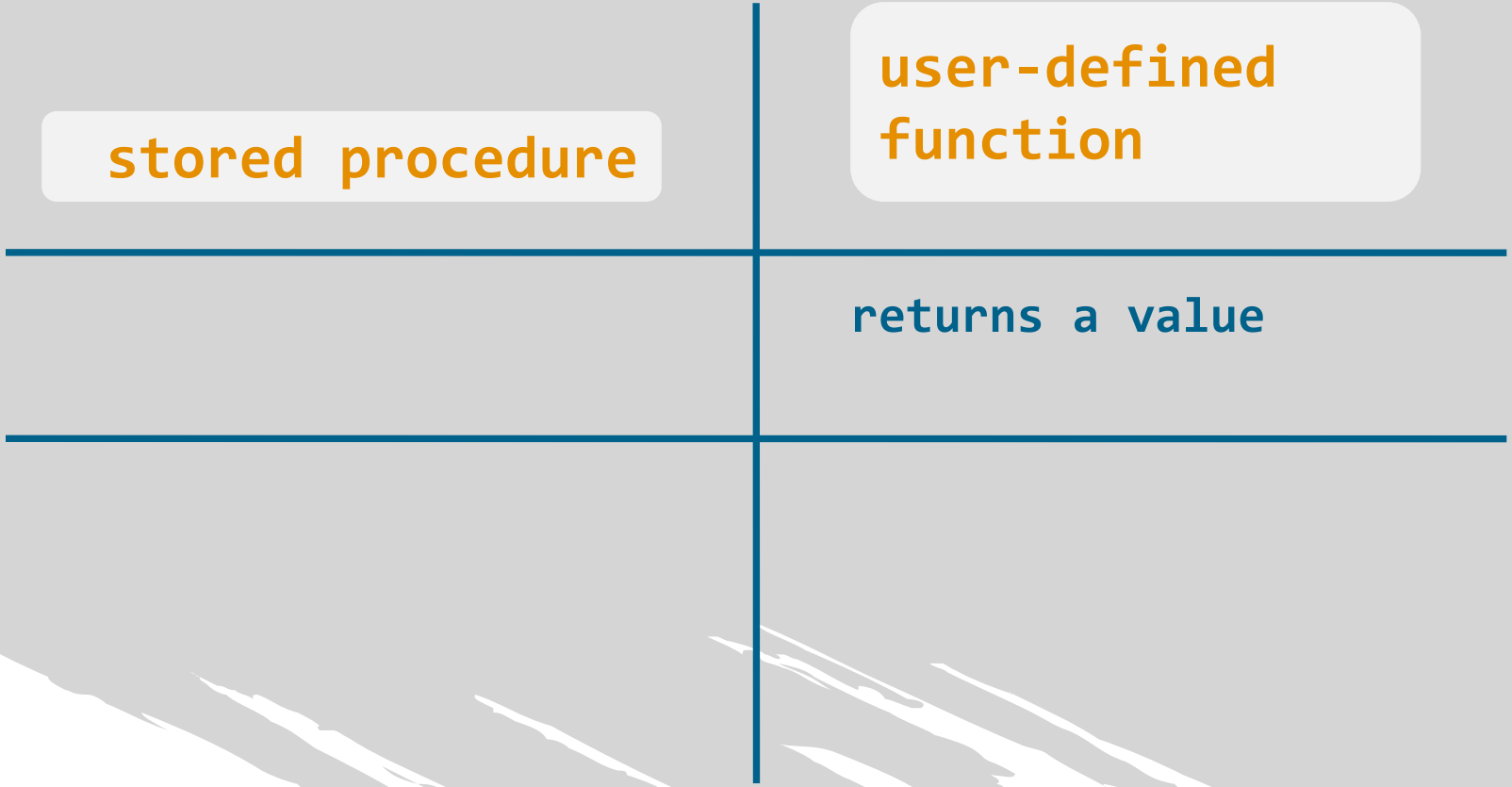
Stored Routines - Conclusion

TECHNICAL DIFFERENCES



Stored Routines - Conclusion

TECHNICAL DIFFERENCES



Stored Routines - Conclusion

TECHNICAL DIFFERENCES

	stored procedure	user-defined function
does not return a value		
returns a value		

Stored Routines - Conclusion

TECHNICAL DIFFERENCES

stored procedure	user-defined function
does not return a value	returns a value
`CALL` procedure;	

Stored Routines - Conclusion

TECHNICAL DIFFERENCES

stored procedure	user-defined function
does not return a value	returns a value
`CALL` procedure;	`SELECT` function;

Stored Routines - Conclusion

CONCEPTUAL DIFFERENCES

Stored Routines - Conclusion

CONCEPTUAL DIFFERENCES

stored procedure

user-defined function

Stored Routines - Conclusion

CONCEPTUAL DIFFERENCES

stored procedure	user-defined function
can have *multiple* OUT parameters	

Stored Routines - Conclusion

CONCEPTUAL DIFFERENCES

stored procedure	user-defined function
can have *multiple* OUT parameters	can return a *single* value only

Stored Routines - Conclusion

CONCEPTUAL DIFFERENCES

stored procedure	user-defined function
can have *multiple* OUT parameters	can return a *single* value only

- if you need to obtain more than one value as a result of a calculation, you are better off *using a procedure*

Stored Routines - Conclusion

CONCEPTUAL DIFFERENCES

stored procedure	user-defined function
can have *multiple* OUT parameters	can return a *single* value only

- if you need to obtain more than one value as a result of a calculation, you are better off *using a procedure*

- if you need to just one value to be returned, then you can *use a function*

Stored Routines - Conclusion

- how about involving an INSERT, an UPDATE, or a DELETE statement?

Stored Routines - Conclusion

- how about involving an INSERT, an UPDATE, or a DELETE statement?

- in those cases, the operation performed will apply changes to the data in your database

Stored Routines - Conclusion

- how about involving an INSERT, an UPDATE, or a DELETE statement?

- in those cases, the operation performed will apply changes to the data in your database

- there will be no value, or values, to be returned and displayed to the user

Stored Routines - Conclusion

CONCEPTUAL DIFFERENCES

stored procedure	user-defined function
can have *multiple* OUT parameters	can return a *single* value only

Stored Routines - Conclusion

CONCEPTUAL DIFFERENCES

stored procedure	user-defined function
can have *multiple* OUT parameters	can return a *single* value only
	INSERT UPDATE DELETE

Stored Routines - Conclusion

CONCEPTUAL DIFFERENCES

stored procedure	user-defined function
can have *multiple* OUT parameters	can return a *single* value only
	INSERT, UPDATE, DELETE

Stored Routines - Conclusion

CONCEPTUAL DIFFERENCES

stored procedure	user-defined function
can have *multiple* OUT parameters	can return a *single* value only
INSERT UPDATE DELETE	INSERT UPDATE DELETE

Stored Routines - Conclusion

CONCEPTUAL DIFFERENCES

stored procedure	user-defined function
can have *multiple* OUT parameters	can return a *single* value only
INSERT UPDATE DELETE	INSERT UPDATE DELETE

Stored Routines - Conclusion

stored procedure

user-defined function

Stored Routines - Conclusion

stored procedure

user-defined function

TECHNICAL DIFFERENCE

Stored Routines - Conclusion

stored procedure	user-defined function

TECHNICAL DIFFERENCE

`CALL` procedure;	`SELECT` function;

Stored Routines - Conclusion

stored procedure	user-defined function

TECHNICAL DIFFERENCE

`CALL` procedure;	`SELECT` function;

CONCEPTUAL DIFFERENCE

Stored Routines - Conclusion

stored procedure	user-defined function

TECHNICAL DIFFERENCE

`CALL` procedure;	`SELECT` function;

CONCEPTUAL DIFFERENCE

	- you can easily include a function as one of the columns inside a SELECT statement

Stored Routines - Conclusion

stored procedure	user-defined function

TECHNICAL DIFFERENCE

`CALL` procedure;	`SELECT` function;

CONCEPTUAL DIFFERENCE

- including a procedure in a SELECT statement is impossible	- you can easily include a function as one of the columns inside a SELECT statement

Stored Routines - Conclusion

- once you become an advanced SQL user, and have gained a lot of practice, you will appreciate the advantages and disadvantages of both types of programs

Stored Routines - Conclusion

once you become an advanced SQL user, and have gained a lot of practice, you will appreciate the advantages and disadvantages of both types of programs

- you will encounter many cases where you should choose between procedures and functions

Stored Routines - Conclusion

- what we did in this section was to lay the foundation of the relevant syntax, as well as performing exercises on the practical aspects of these tools