# MMSQL

# Nasrin Khodapanah



## AGENDA

- Security
  - Stored Procedures
  - Stored Functions
  - Triggers



# **Stored Procedures**



```
mysql> DELIMITER //
mysql> CREATE PROCEDURE procedure_name(parameter_1, parameter_2, . . ., parameter_n)
mysql> BEGIN
mysql> instruction_1;
mysql> instruction_2;
mysql> . . .
mysql> instruction_n;
mysql> END //
mysql> DELIMITER;
```

#### CREATE PROCEDURE

```
procedure_name([parameter1 [,
parameter2, ...]])
```

Body of procedure;



CREATE PROCEDURE show\_race\_query()
SELECT id, name, species\_id, price
FROM race;



```
CREATE PROCEDURE show_race_query()
BEGIN

SELECTid, name, species_id, price
FROM race;
END;
```



# STORED PROCEDURES - DELIMITER

BY DEFAULT = ;

TO CHANGE:

DELIMITER

**SELECT** 'test' |



## **STORED PROCEDURES - DELIMITER**

```
DELIMITER
CREATE PROCEDURE show_race()
BEGIN
  SELECT id, name, species_id AS 'Species id', price
  FROM race;
END
DELIMITER;
```

CALL show\_race;



IN: It is a parameter whose value is supplied to the stored procedure. This value will be used during the procedure (for a calculation or selection for example).

OUT: It is a parameter whose value will be determined during the procedure and can be used outside it.

**INOUT:** This parameter is used during the procedure, see if it's changed by it, and will then be usable outside.



## STORED PROCEDURES - PARAMETERS - SYNTAX

- Direction: IN, OUT, INOUT
- Name: name of parameters
- Type: datatype



```
CREATE PROCEDURE show_race_by_species(IN
 p_species_id INT)
BEGIN
  SELECT id, name, species_id, price
  FROM race
     WHERE species_id = p_species_id;
END
DELIMITER;
```



## STORED PROCEDURES — PARAMETERS - SYNTAX

```
CALL show_race_by_species (1);
```

CALL show\_race\_by\_species (@species\_id);



```
DELIMITER
CREATE PROCEDURE count_race_by_species
 (IN p_species_id INT, OUT p_num_races INT)
BEGIN
SELECT COUNT(*) INTO p_num_races
  FROM race
  WHERE species_id = p_species_id;
END
DELIMITER;
```



## STORED PROCEDURES — PARAMETERS - SYNTAX

SELECT id, name INTO @var1, @var2 FROM animal WHERE id = 7; SELECT @var1, @var2;



```
CALL count_race_by_species(2,
    @num_cat_race);
```

SELECT @num\_cat\_race;



```
DELIMITER
CREATE PROCEDURE calculate_price(IN p_a nimal_id INT,
  INOUT p_price DECIMAL(7,2))
BEGIN
  SELECT p_price + COALESCE(race.price,
  species.price) INTO p_price
   FROM animal
   INNER JOIN species
      ON species.id = animal.species id
   LEFT JOIN race
      ON race.id = animal.race id
  WHERE animal.id = p_animal_id;
END |
```

```
\mathbf{SET} @price = 0;
CALL calculate price (13, @price);
SELECT @price AS first total;
CALL calculate_price (24, @price);
SELECT @price AS second total;
CALL calculate_price (42, @price);
SELECT @price AS third_total;
CALL calculate_price (75, @price);
SELECT @price AS grand total;
```



#### STORED PROCEDURES – EXTRA COMMANDS

SHOW PROCEDURE STATUS; -- show all the created store procedures

SHOW CREATE PROCEDURE show\_race;—show a stored procedure definition

DROP PROCEDURE show\_race; -- delete a stored
procedure



## Tools

- Local variable which will allow to store and change values in the course of a procedure
- Conditions which will allow to perform certain instructions only if a certain condition is met.
- Loops which will allow to repeat a command several times.



## **STORED PROCEDURES - VARIABLES**

DECLARE variable\_name variable\_type [DEFAULT default\_value];

#### **BEGIN**

- variable declarations
- instructions

## END;

See today\_tomorrow procedure!



## **STORED PROCEDURES - VARIABLES**

A variable whose name begins with the @ sign is a session variable. It is available and accessible until the session ends.

See the **test** procedure!

A local variable only exists in the instruction block they're declared.

When END is reached, all variables are destroyed.



## **STORED PROCEDURES - VARIABLES**

A local variable also exists for a nested block.

Two local variables can have the same name as long they're in different blocks.

See test\_range1 procedure!



## **IF CASE:**

If something, then do something.

IF condition THEN instructions

END IF;



#### **IF ELSE CASE:**

If something, then do something else do another thing.

IF condition THEN instructions

**ELSE** 

instructions

END IF;

See is\_adopted procedure;



## IF ELSE IF CASE:

If something, then do something else if something else do something else Else do otherthing.

Fcondition THEN

instruction

**ELSE IF** condition THEN

instruction

ELSE

condition

END I;

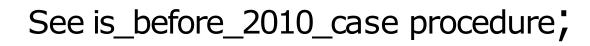
See is\_before\_2010 procedure;



#### **SWITCH CASE:**

```
CASE value_to_compare
   WHEN possibility1 THEN
   instructions
   [WHEN possibility2 THEN
   instructions]
   ELSE
   instructions]
```

END CASE;





## STORED PROCEDURES — LOOPS

WHILE;

WHILE condition DO instructions

END WHILE;

See count\_in\_while function;



## STORED PROCEDURES — LOOPS

REPEAT;

Opposite of WHILE – execute the instruction until the given condition becomes true.

REPEAT condition

**UNTIL** instruction

**END** REPEAT;

See count\_in\_repeat procedure;

## STORED PROCEDURES — LOOPS

If condition is false from the beginning, then, the loop won't be executed at all for while loops.

CALL count\_in\_while(0);

If condition is false from the beginning, then, the loop will be executed only once for repeat loops.

CALL count\_in\_repeat(0);

