MMSQL

Nasrin Khodapanah

Agenda

- Security
  - Stored Procedures
  - Stored Functions
  - Triggers

Stored Procedures

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;

Stored Procedures

```
CREATE PROCEDURE show_race_query()
SELECT id, name, species_id, price
FROM race;
```

Stored Procedures

```
CREATE PROCEDURE show_race_query()
BEGIN
    SELECT id, name, species_id, price
    FROM race;
END;
```

HERZING COLLEGE

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;
```

Stored Procedures - parameters

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

Stored Procedures – parameters

```
DELIMITER |
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 ;
```

HERZING COLLEGE

Stored Procedures – parameters - syntax

```
CALL show_race_by_species (1);
SET @species_id:= 2;
CALL show_race_by_species (@species_id);
```

Stored Procedures – parameters

```
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 ;
```

HERZING COLLEGE

Stored Procedures – parameters - syntax

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

Stored Procedures – parameters

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

SELECT @num_cat_race;
```

HERZING COLLEGE

Stored Procedures – parameters

```
DELIMITER |
CREATE PROCEDURE calculate_price(IN p_animal_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 |
```

Stored Procedures – parameters

```
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

Stored Procedures

- 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!

HERZING COLLEGE

Stored Procedures – condition

IF CASE:

If something, then do something.

```
IF condition THEN
    instructions
END IF;
```

HERZING COLLEGE

Stored Procedures – condition

IF ELSE CASE:

If something, then do something else do another thing.

```
IF condition THEN
instructions
ELSE
instructions
END IF;
```

See is_adopted procedure;

Stored Procedures – condition

IF ELSE IF CASE:

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

```
IF condition THEN
instruction
ELSE IF condition THEN
instruction
ELSE
condition
END IF;
```

See is_before_2010 procedure;

HERZING COLLEGE

Stored Procedures – condition

SWITCH CASE:

```
CASE value_to_compare
    WHEN possibility1 THEN
    instructions
    [WHEN possibility2 THEN
    instructions]
  ...
    [ELSE
    instructions]
END CASE;
```

See is_before_2010_case procedure;

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);
```