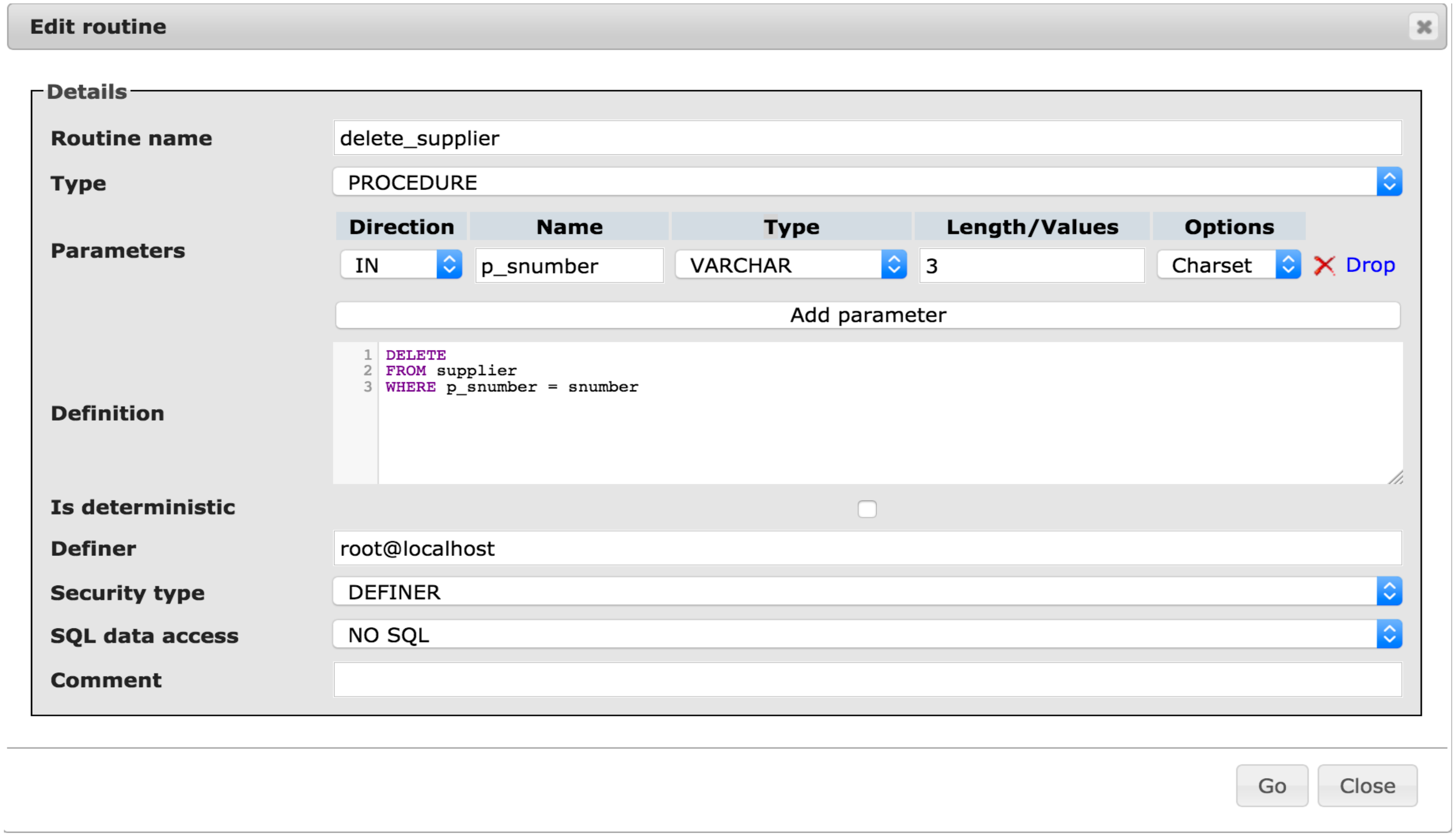
Stored procedures

A stored procedure is a certain piece of code (the procedure) consisting of declarative and procedural SQL statements stored in the catalog of a database that can be activated by calling it from a program, a trigger, or another stored procedure.

SQL statements, such as CREATE, UPDATE, and SELECT, possibly complemented with procedural statements, such as IF-THEN-ELSE and WHILE-DO.

Create a procedure

Click Routines > add routine



Edit routine

Details

Routine name	delete_supplier
Type	PROCEDURE
Is deterministic	☐
Definer	root@localhost
Security type	DEFINER
SQL data access	NO SQL
Comment	

Parameters

Direction	Name	Type	Length/Values	Options
IN	p_snumber	VARCHAR	3	Charset

Definition

```
DELETE
FROM supplier
WHERE p_snumber = snumber
```

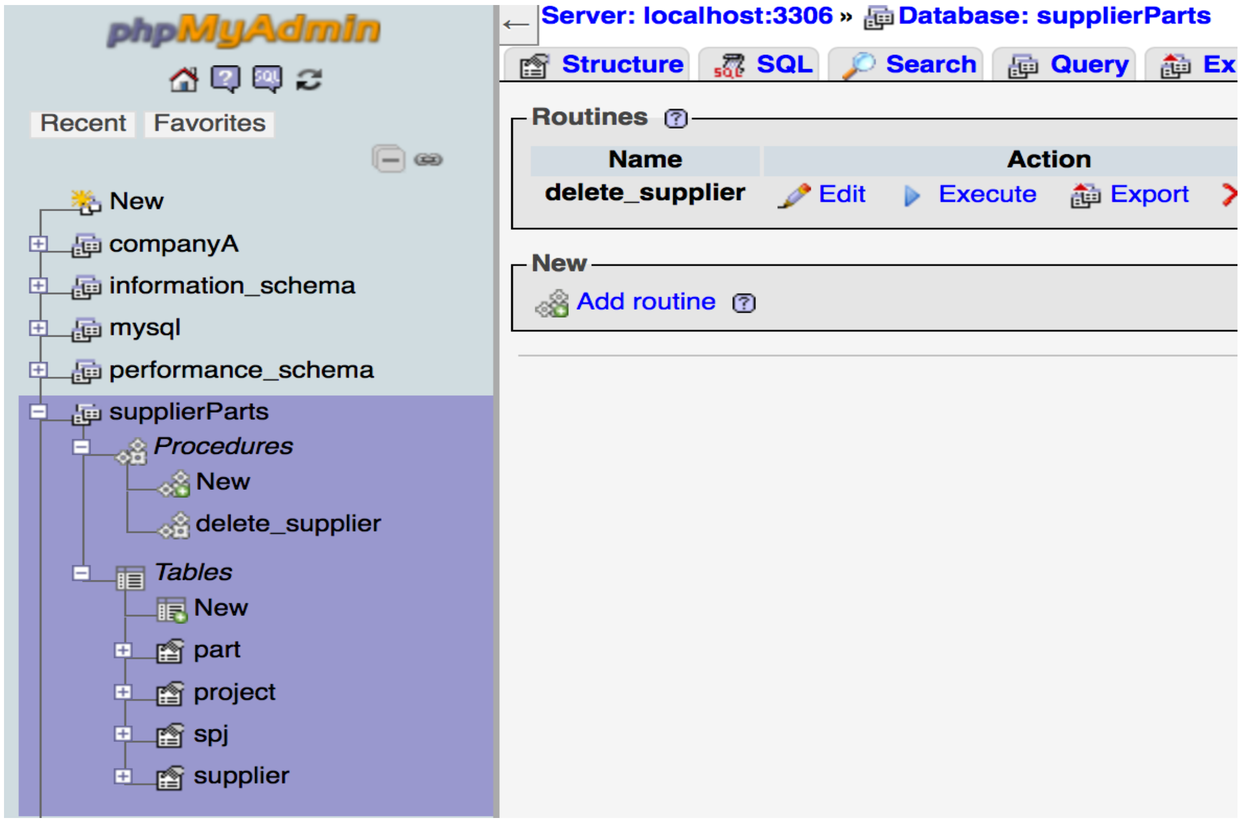
Create procedure statement

DEFINITION

```
<create procedure statement> ::=
   CREATE PROCEDURE <procedure name> ( [ <parameter list> ] )
      <routine body>

<parameter list> ::=
   <parameter specification> [ , <parameter specification> ]...

<parameter specification> ::=
   [ IN | OUT | INOUT ] <parameter> <data type>
```

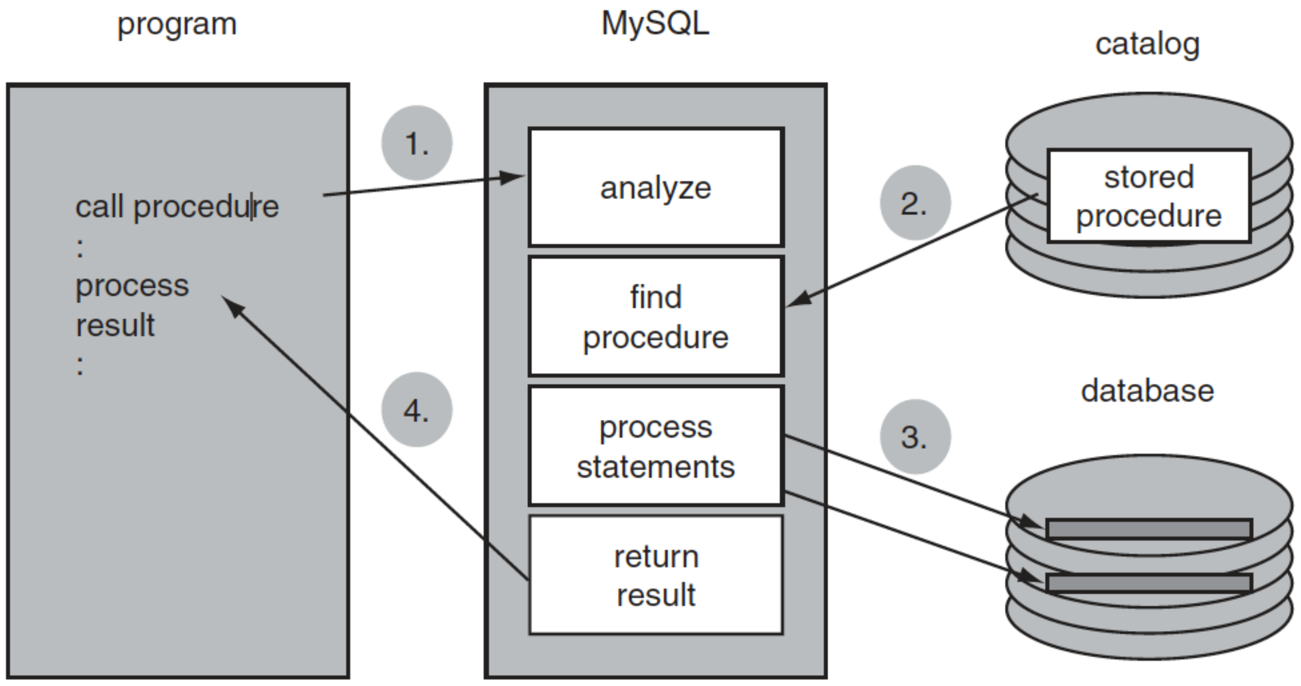
Call a procedure directly or click execute

```
SET @p0='s6';
CALL delete_supplier(@p0);
```



Name	Action
delete_supplier	Edit Execute Export

Processing steps of stored procedure



Parameters

A stored procedure has zero, one, or multiple parameters.

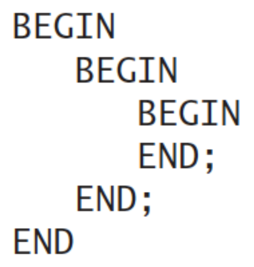
Three types of parameters are supported:

Input parameters, data can be passed to a stored

Output parameters when an answer or result must be returned.

Input/Output parameters, can act as an input as well as an output parameter.

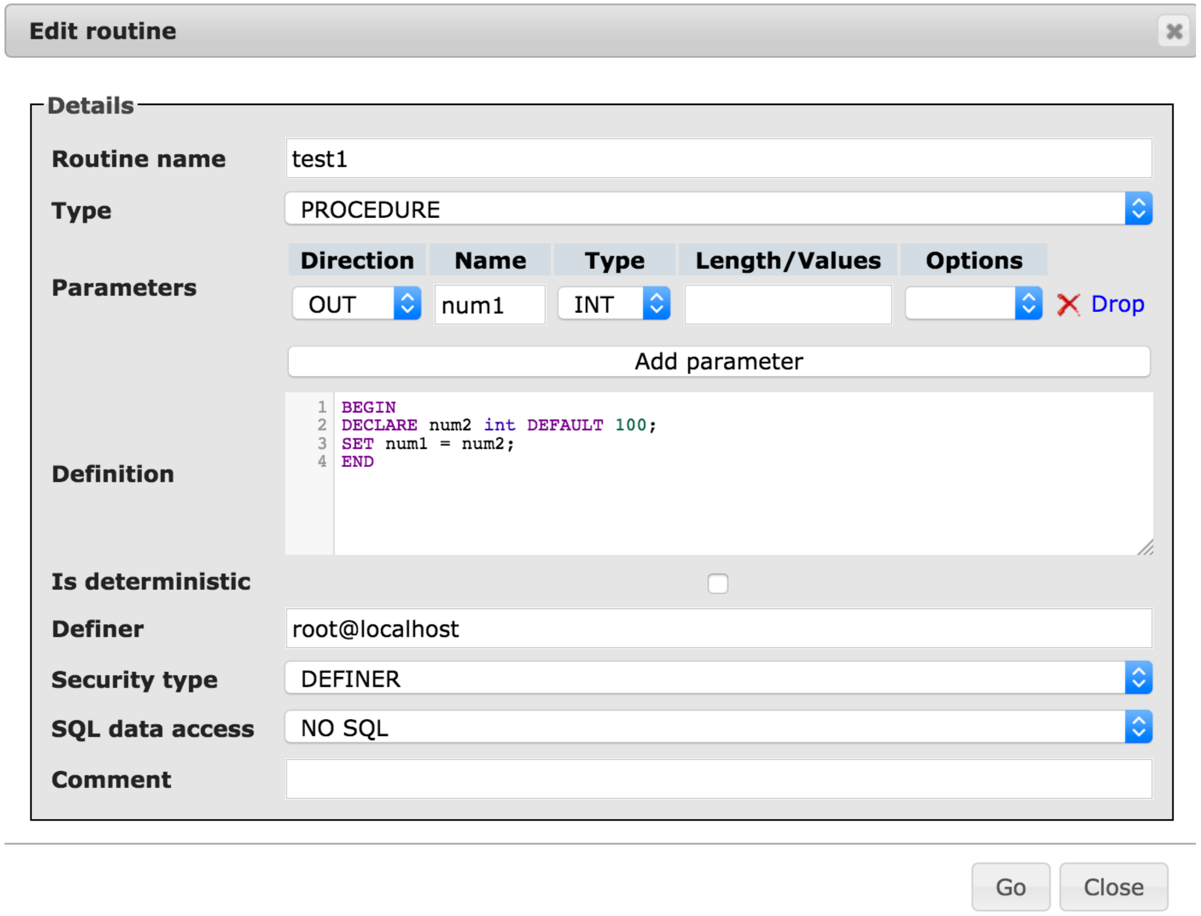
BEGIN END block



```
BEGIN
   BEGIN
      BEGIN
      END;
   END;
END
```

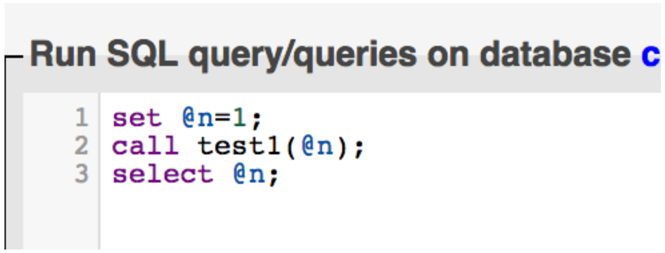
Local variables

```
<declare variable statement> ::=
   DECLARE <local variable list> <data type>
      [ DEFAULT <scalar expression> ]
```



Routine name	test1
Type	PROCEDURE
Parameters	Direction: OUT; Name: num1; Type: INT; Length/Values: ; Options:
Definition	(below)
Is deterministic	☐
Definer	root@localhost
Security type	DEFINER
SQL data access	NO SQL
Comment	

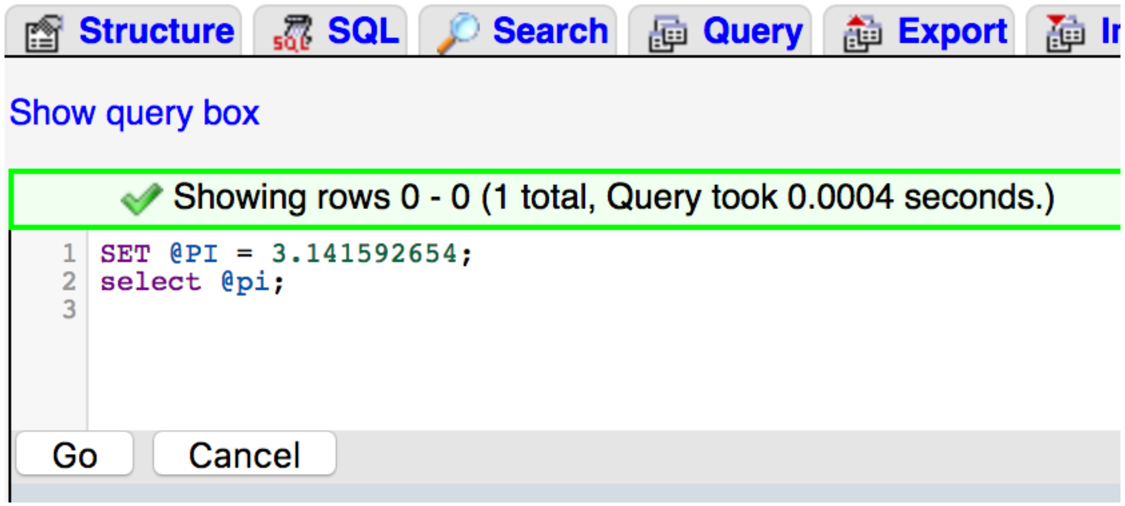
```
BEGIN
DECLARE num2 int DEFAULT 100;
SET num1 = num2;
END
```



```
set @n=1;
call test1(@n);
select @n;
```

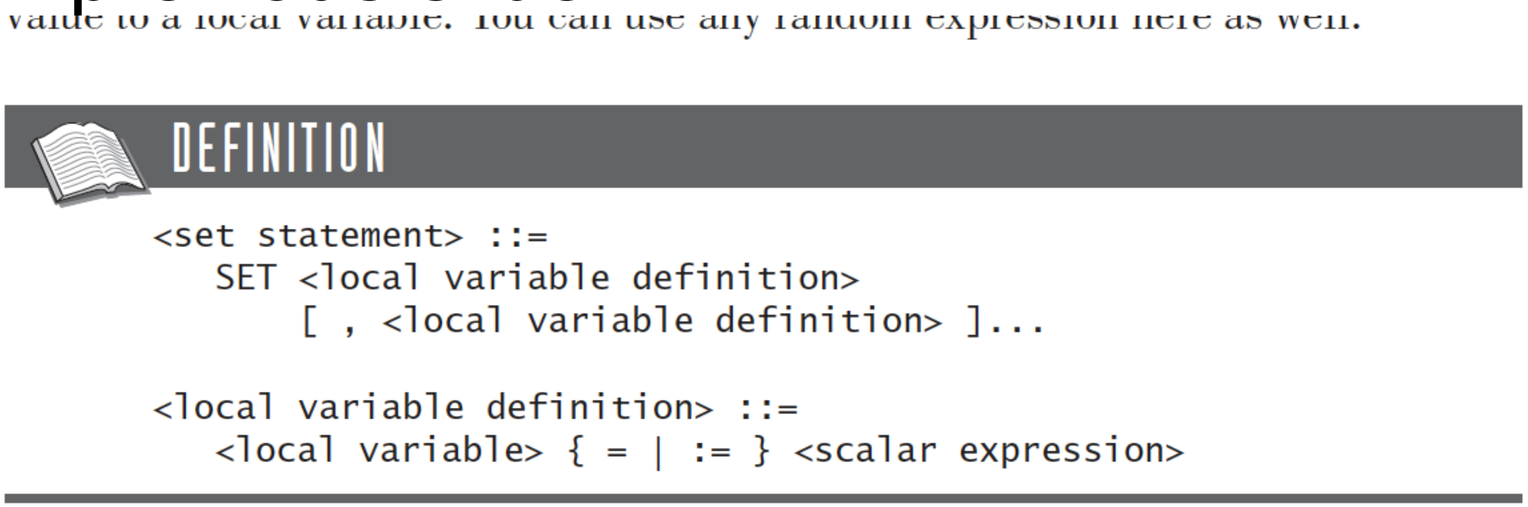
Set Statement for User variables
Set statment for local variables

- **User variables,** defined outside a stored procedure must start with @.
- **Local variables** in previous slide.



Showing rows 0 - 0 (1 total, Query took 0.0004 seconds.)

```
SET @PI = 3.141592654;
select @pi;
```

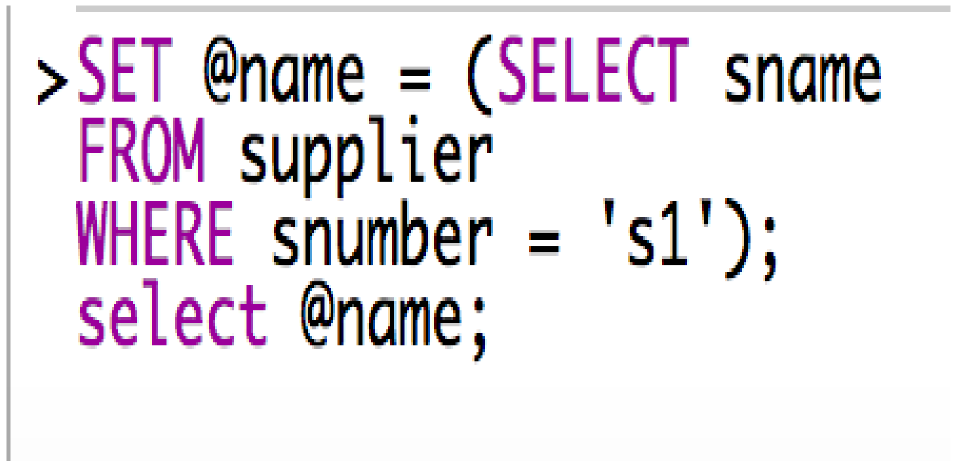


value to a local variable. You can use any random expression here as well.

DEFINITION

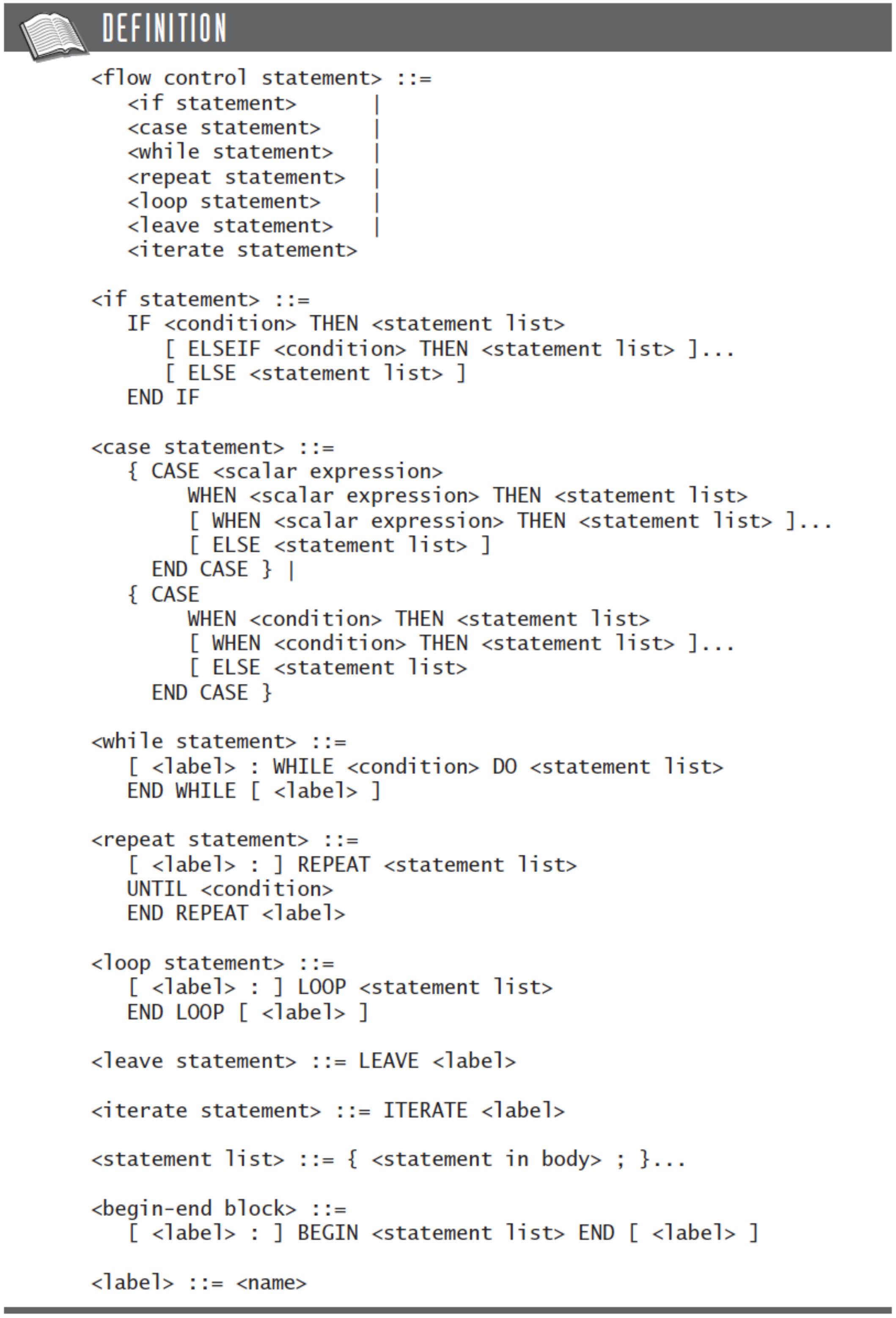
```
<set statement> ::=
   SET <local variable definition>
       [ , <local variable definition> ]...

<local variable definition> ::=
   <local variable> { = | := } <scalar expression>
```

Example-- User variable @sname



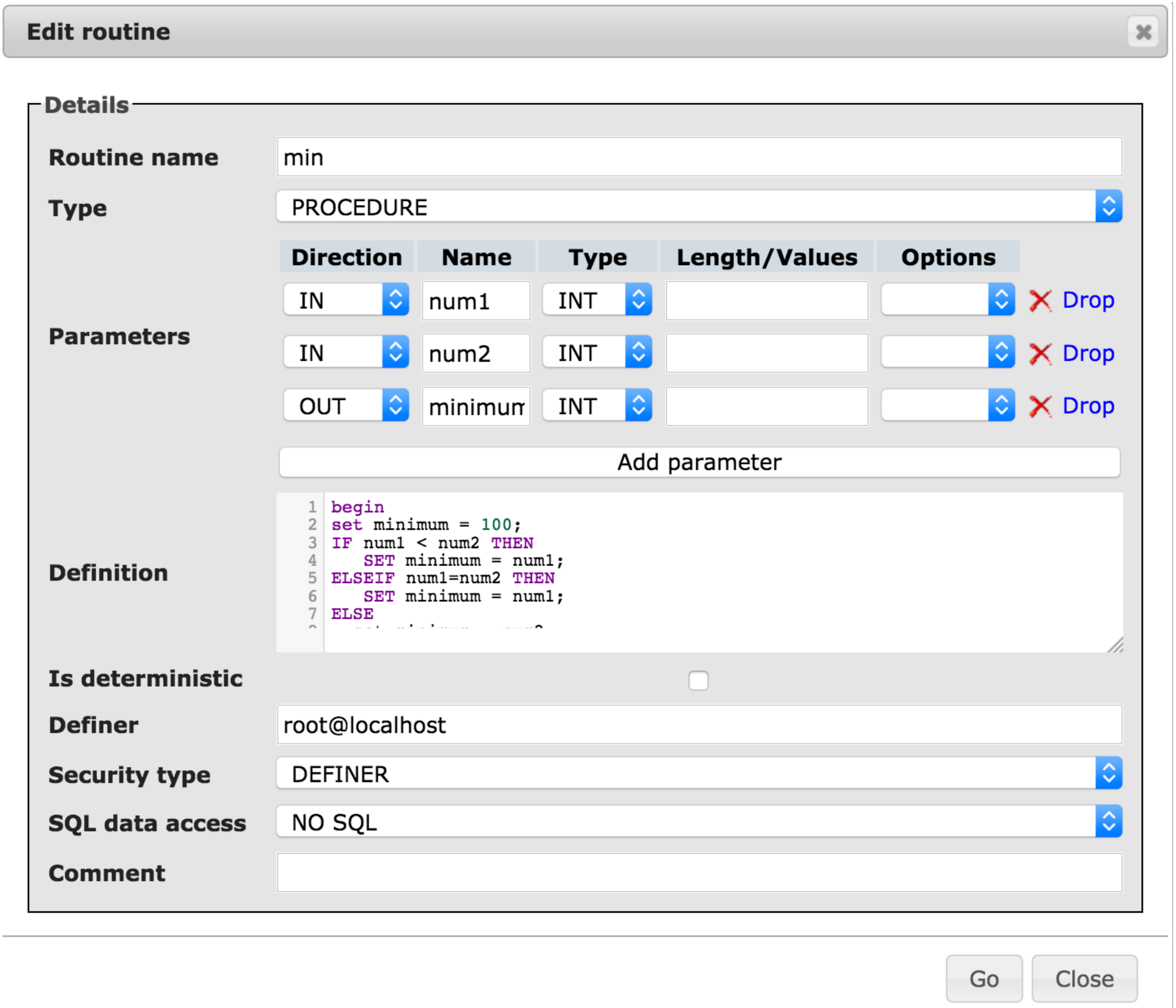
```
>SET @name = (SELECT sname
 FROM supplier
 WHERE snumber = 's1');
 select @name;
```

Flow control



DEFINITION

```
<flow control statement> ::=
   <if statement>      |
   <case statement>    |
   <while statement>   |
   <repeat statement>  |
   <loop statement>    |
   <leave statement>   |
   <iterate statement>

<if statement> ::=
   IF <condition> THEN <statement list>
      [ ELSEIF <condition> THEN <statement list> ]...
      [ ELSE <statement list> ]
   END IF

<case statement> ::=
   { CASE <scalar expression>
        WHEN <scalar expression> THEN <statement list>
        [ WHEN <scalar expression> THEN <statement list> ]...
        [ ELSE <statement list> ]
     END CASE } |
   { CASE
        WHEN <condition> THEN <statement list>
        [ WHEN <condition> THEN <statement list> ]...
        [ ELSE <statement list>
     END CASE }

<while statement> ::=
   [ <label> : WHILE <condition> DO <statement list>
   END WHILE [ <label> ]

<repeat statement> ::=
   [ <label> : ] REPEAT <statement list>
   UNTIL <condition>
   END REPEAT <label>

<loop statement> ::=
   [ <label> : ] LOOP <statement list>
   END LOOP [ <label> ]

<leave statement> ::= LEAVE <label>

<iterate statement> ::= ITERATE <label>

<statement list> ::= { <statement in body> ; }...

<begin-end block> ::=
   [ <label> : ] BEGIN <statement list> END [ <label> ]

<label> ::= <name>
```


Edit routine

Details

Routine name	min
Type	PROCEDURE

Parameters

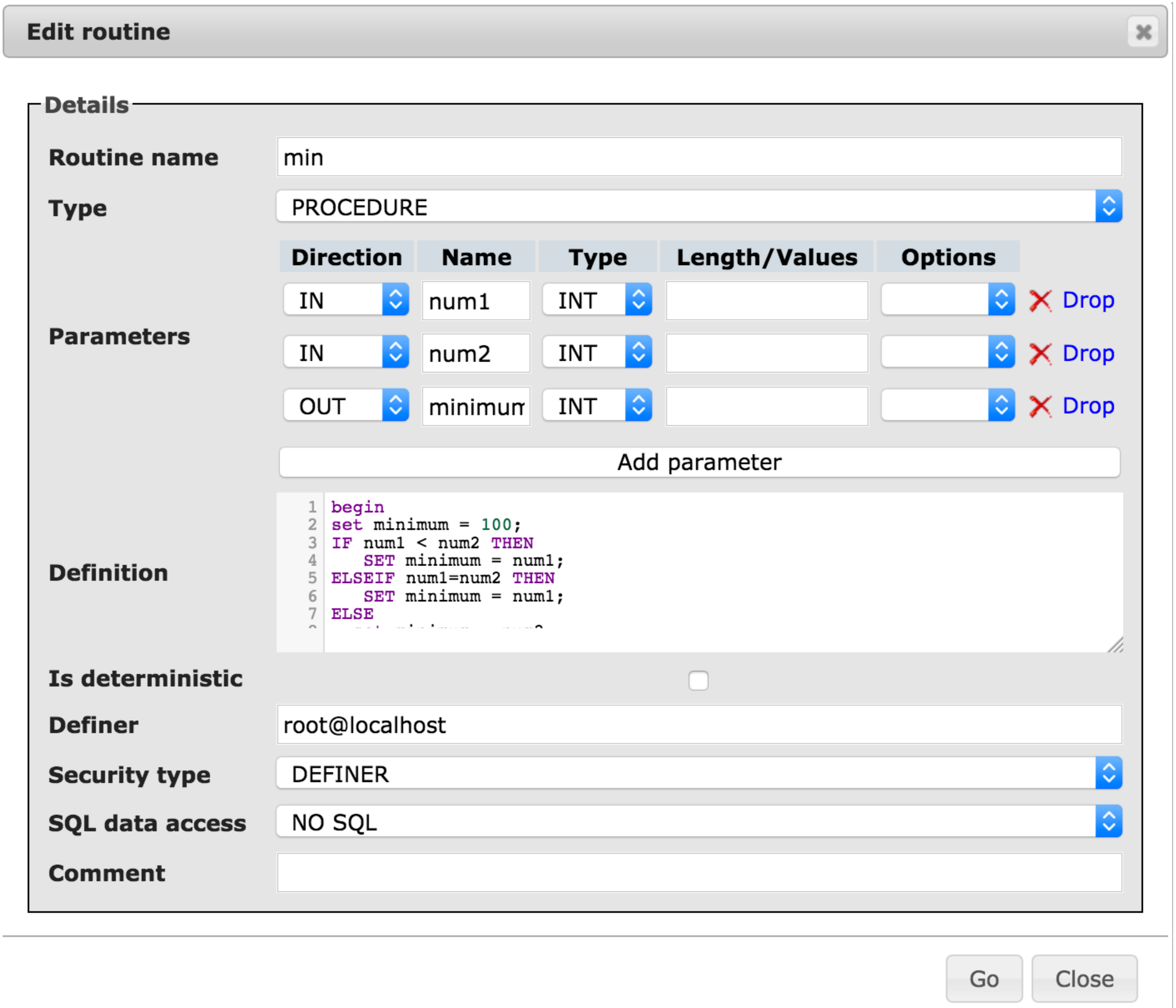
Direction	Name	Type	Length/Values	Options	
IN	num1	INT			Drop
IN	num2	INT			Drop
OUT	minimum	INT			Drop

Add parameter

Definition

```
begin
set minimum = 100;
IF num1 < num2 THEN
    SET minimum = num1;
ELSEIF num1=num2 THEN
    SET minimum = num1;
ELSE
```

Is deterministic	☐
Definer	root@localhost
Security type	DEFINER
SQL data access	NO SQL
Comment	



Details

Routine name	min
Type	PROCEDURE

Parameters

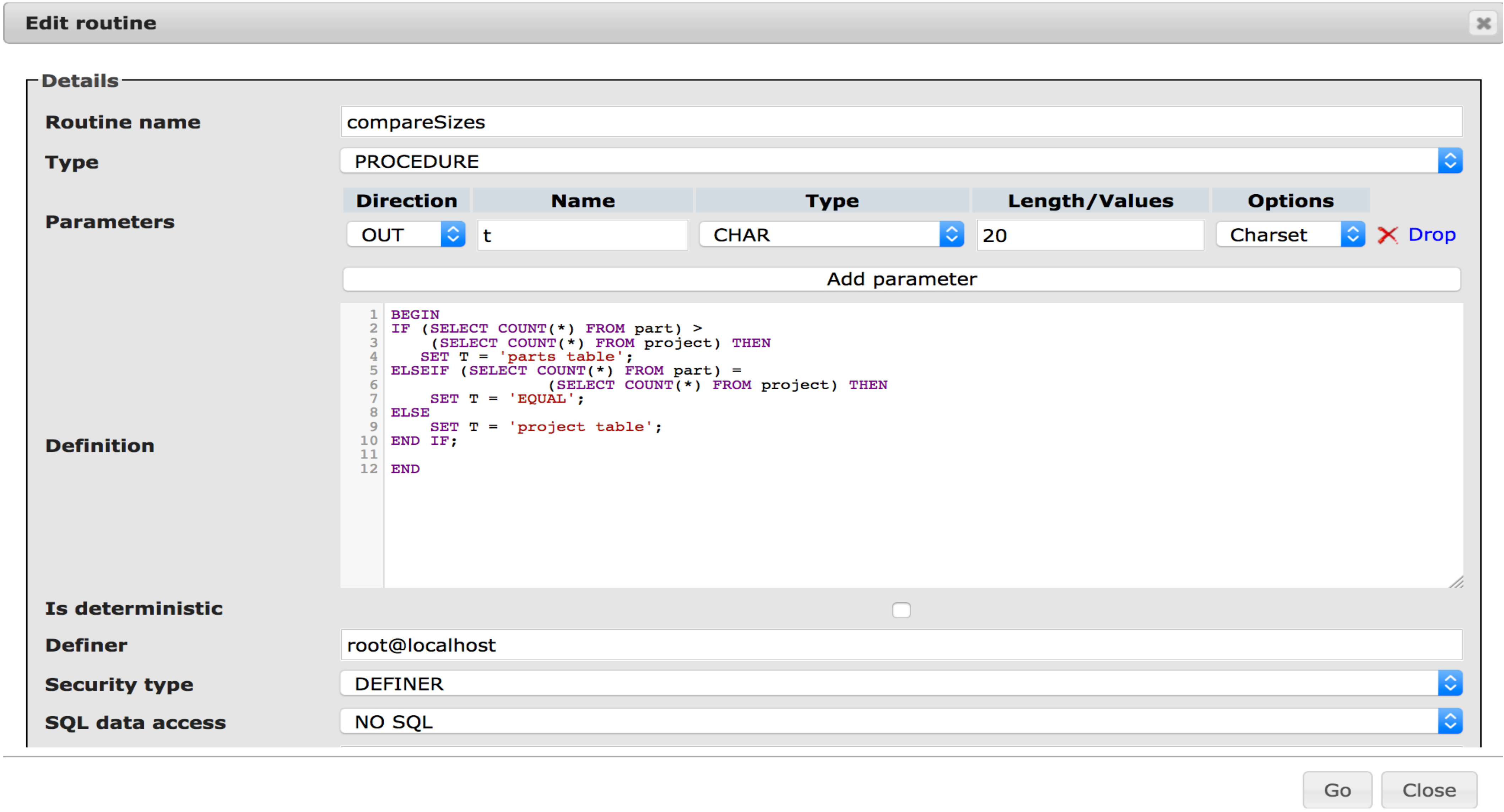
Direction	Name	Type	Length/Values	Options	
IN	num1	INT			Drop
IN	num2	INT			Drop
OUT	minimum	INT			Drop

Definition

```
begin
set minimum = 100;
IF num1 < num2 THEN
    SET minimum = num1;
ELSEIF num1=num2 THEN
    SET minimum = num1;
ELSE
```

Is deterministic	☐
Definer	root@localhost
Security type	DEFINER
SQL data access	NO SQL
Comment	

Using SQL in stored procedures.
Example: Find max number of rows.



Edit routine

Details

Routine name	compareSizes
Type	PROCEDURE

Parameters

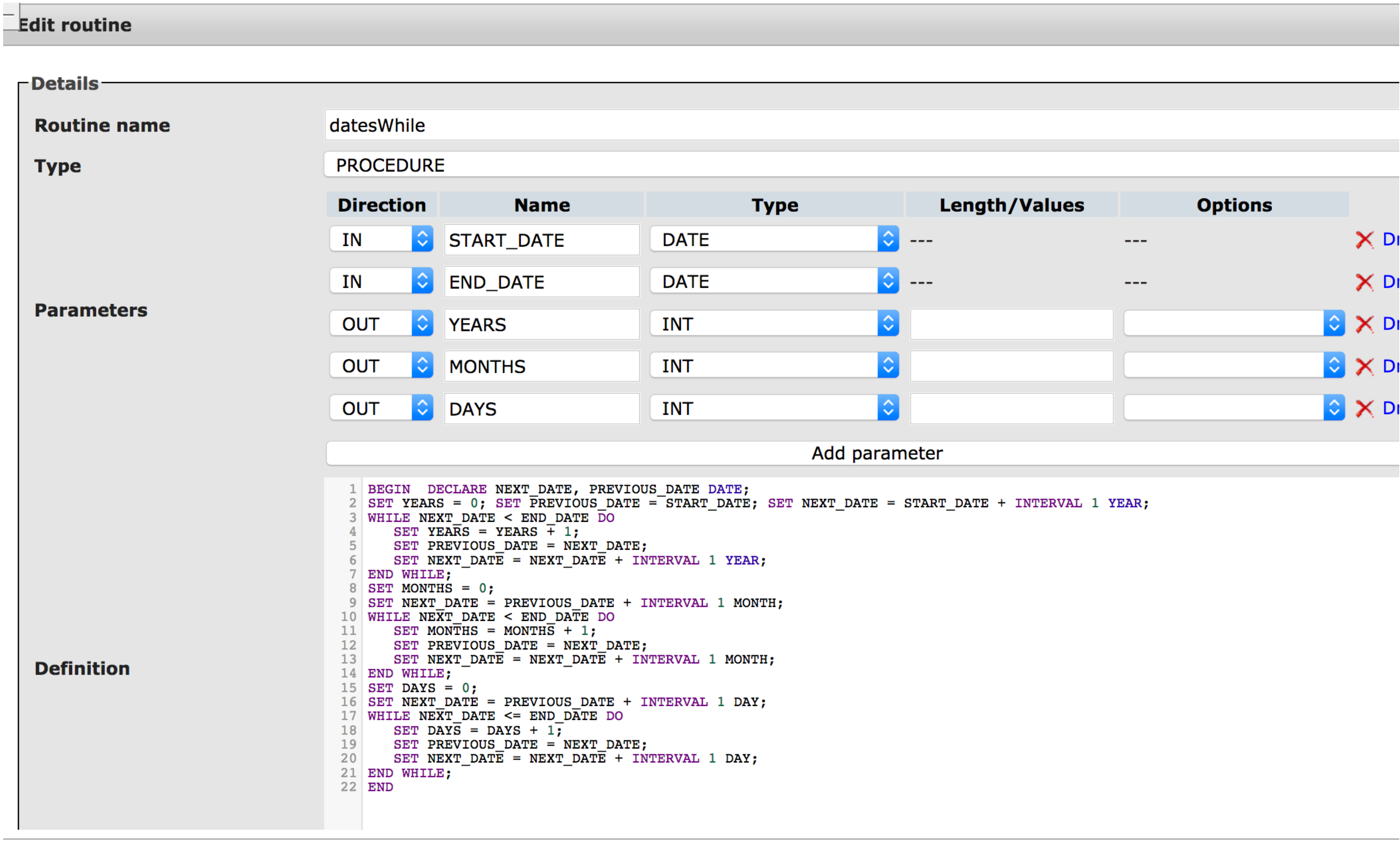
Direction	Name	Type	Length/Values	Options
OUT	t	CHAR	20	Charset

Definition

```
BEGIN
IF (SELECT COUNT(*) FROM part) >
    (SELECT COUNT(*) FROM project) THEN
   SET T = 'parts table';
ELSEIF (SELECT COUNT(*) FROM part) =
            (SELECT COUNT(*) FROM project) THEN
    SET T = 'EQUAL';
ELSE
    SET T = 'project table';
END IF;

END
```

Is deterministic	☐
Definer	root@localhost
Security type	DEFINER
SQL data access	NO SQL

While
Find difference between 2 dates



Edit routine

Details

Routine name	datesWhile
Type	PROCEDURE

Parameters

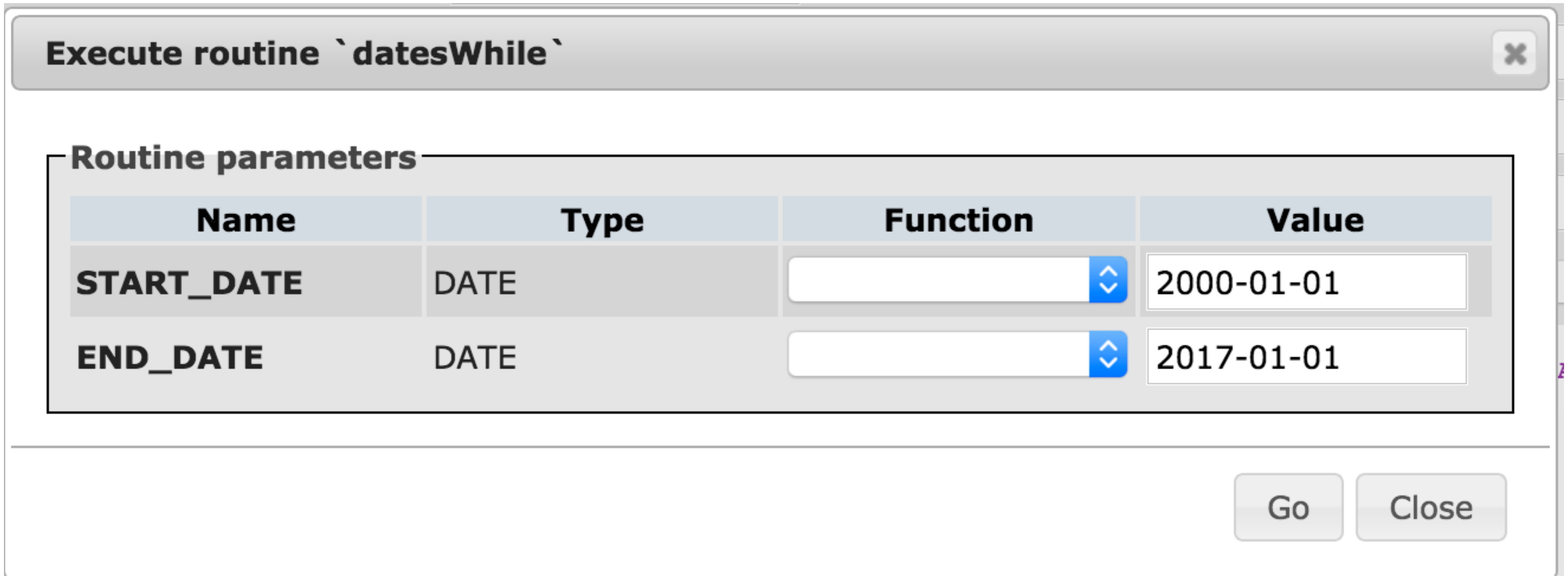
Direction	Name	Type	Length/Values	Options
IN	START_DATE	DATE	---	---
IN	END_DATE	DATE	---	---
OUT	YEARS	INT		
OUT	MONTHS	INT		
OUT	DAYS	INT		

Add parameter

Definition

```
BEGIN  DECLARE NEXT_DATE, PREVIOUS_DATE DATE;
SET YEARS = 0; SET PREVIOUS_DATE = START_DATE; SET NEXT_DATE = START_DATE + INTERVAL 1 YEAR;
WHILE NEXT_DATE < END_DATE DO
    SET YEARS = YEARS + 1;
    SET PREVIOUS_DATE = NEXT_DATE;
    SET NEXT_DATE = NEXT_DATE + INTERVAL 1 YEAR;
END WHILE;
SET MONTHS = 0;
SET NEXT_DATE = PREVIOUS_DATE + INTERVAL 1 MONTH;
WHILE NEXT_DATE < END_DATE DO
    SET MONTHS = MONTHS + 1;
    SET PREVIOUS_DATE = NEXT_DATE;
    SET NEXT_DATE = NEXT_DATE + INTERVAL 1 MONTH;
END WHILE;
SET DAYS = 0;
SET NEXT_DATE = PREVIOUS_DATE + INTERVAL 1 DAY;
WHILE NEXT_DATE <= END_DATE DO
    SET DAYS = DAYS + 1;
    SET PREVIOUS_DATE = NEXT_DATE;
    SET NEXT_DATE = NEXT_DATE + INTERVAL 1 DAY;
END WHILE;
END
```

Execute datesWhile



Execute routine `datesWhile`

Routine parameters

Name	Type	Function	Value
START_DATE	DATE		2000-01-01
END_DATE	DATE		2017-01-01