



Databasteknik

Aggregerade, Subqueries, SP

Utbildare: Mikael Olsson

mikael.olsson@emmio.se

076-174 90 43

NACKADEMIN

Socrative

<https://www.socrative.com/>

- Frågehanterare
 - Logga in som student
 - Ange rum “Emmio”
 - Få upp en vänta-skärm
 - Quiz: Join & Views



Waiting for the next activity to begin...

Aggregerade funktioner

- AVG
- COUNT
- SUM
- MIN
- MAX
- GROUP_CONCAT
 - DISTINCT
 - ORDER BY
 - SEPARATOR

Sträng-funktioner

- CONCAT
- LENGTH
- LEFT
- REPLACE
- SUBSTRING
- TRIM
- FORMAT

Datum-funktioner

- CURDATE
 - DATEDIFF
 - DAY
 - DATE_ADD
 - DATE_SUB
 - DATE_FORMAT
 - DAYNAME
 - DAYOFWEEK
- NOW
 - MONTH
 - STR_TO_DATE
 - WEEK
 - WEEKDAY
 - YEAR

Subqueries

- *Predicate subqueries* - utökade logiska konstruktioner i WHERE- och HAVING-delarna.
- *Skalära subqueries* - fristående frågor som returnerar ett enda värde.

Predicate subqueries

- Kan enbart användas i WHERE- och HAVING-delarna. Måste returnera en kolumn.

```
mysql> SELECT * FROM Client WHERE C_ID IN  
      -> (SELECT C_ID FROM Products);
```

C_ID	Name	City
1	A K Ltd	Delhi
2	V K Associate	Mumbai
3	R K India	Banglore
5	A T Ltd	Delhi

```
4 rows in set (0.00 sec)
```

Skalära subqueries

- Returnerar ett enda värde.
- Kan användas där en kolumn kan användas.

```
1 SELECT
2     c.customerName,
3     (SELECT SUM(amount)
4      FROM payments
5      WHERE customerNumber = c.customerNumber) AS TotalPaid
6 FROM customers c;
```

Overview Output Snippets Result (1) ×

← → ↗ ⌕ 🔍 📄 🔄 🔍

customerName TotalPaid

Atelier graphique	22314.36
Signal Gift Stores	80180.98
Australian Collectors, Co.	180585.06999999998
La Rochelle Gifts	116949.68000000001
Baane Mini Imports	104224.79

Fetches 122 records. Duration: 0.000 seconds

HAVING

- Hur visar vi enbart rader där avgQuantityInStock > 3500?

```
1 SELECT productLine, AVG(quantityInStock) AS avgQuantityInStock
2 FROM products
3 GROUP BY productLine;
```

productLine	avgQuantityInStock
Classic Cars	5767.9737
Motorcycles	5338.5385
Planes	5190.5833
Ships	2981.4444
Trains	5565.3333
Trucks and Buses	3259.1818
Vintage Cars	5203.3333

HAVING

WHERE funkar inte i aggregerade funktioner

```
1 SELECT productLine, AVG(quantityInStock) AS avgQuantityInStock
2 FROM products
3 WHERE avgQuantityInStock > 3500
4 GROUP BY productLine;
```

249 22:12:10 Error Code: 1054Unknown column 'avgQuantityInStock' in 'where clause'

```
1 SELECT productLine, AVG(quantityInStock) AS avgQuantityInStock
2 FROM products
3 WHERE AVG(quantityInStock) > 3500
4 GROUP BY productLine;
```

250 22:14:26 Error Code: 1111Invalid use of group function

HAVING

```
1 • SELECT productLine, AVG(quantityInStock) AS avgQuantityInStock
2 FROM products
3 GROUP BY productLine
4 HAVING avgQuantityInStock > 3500;
```

Result (1)	Result (1)	Result (1)	Result (1)	Result (1)	Result (1)	Result (1)	Result (1)
						<input type="text"/>	Fetches 5 records. Duration: 0.00

productLine	avgQuantityInStock
Classic Cars	5767.9737
Motorcycles	5338.5385
Planes	5190.5833
Trains	5565.3333
Vintage Cars	5203.3333

HAVING

- WHERE används för att begränsa rader.
 - Används även för att avgöra vilka tabeller och index som ska användas.
- HAVING är ett “filter” på resultatet
 - Lägggs på efter ORDER BY och GROUP BY.
- WHERE ger bättre performance än HAVING.

Stored Procedures

- Batches
- Stored procedures
- Lokala och globala variabler
- Parametrar

Stored Procedures

Procedurkod

- Procedurkod kan lagras i "stored procedures" på servern.
- Procedurkod kan också skickas från klienten till servern för att köras där.

Stored Procedures

Batch

- En batch är två eller flera SQL-satser
 - Skickas ihop, som ett nätverkspaket.
 - Reducerar antalet anrop
- Analyseras tillsammans.

```
SELECT * FROM products;  
SELECT * FROM orders;
```

USE

- `USE databasnamn`
 - USE ändrar aktuell databas
 - Som att dubbelklicka i WorkBench
 - Exempel på hur man ändrar databas i en batch:

```
USE classicmodels;  
SELECT * FROM products;
```

```
USE komplit_ikt;  
SELECT * FROM amne;
```

Regler för batchar

- Vissa satser måste ligga i en egen batch.
 - T ex CREATE PROCEDURE
- För att köra en lagrad procedur använder man CALL.
 - I MSSQL: EXEC
- Man kan inte ta bort och återskapa tabell i samma batch.
- Lokala variabler gäller bara i en batch.

Skript

- Ett skript är en batch som lagras i en fil.
- Kan köras från Workbench eller kommandoraden.
- Det är vanligt att använda ett skript för att skapa en databas, dess tabeller och constraints.
- Exempel är *classicmodels*.

Stored procedure

- En lagrad procedur är förkompilerad och optimerad sql som sparas på servern.

Stored procedure

Fördelar

- Snabbare exekvering
 - Förkompilerade, optimerade, cachade i servern.
- Reducerad nätverkstrafik
 - SQL skickas inte längre över nätet.
- Bättre generalitet
 - Parametrar ger mångsidig kod
- Bättre säkerhet
 - Man kan tillåta exekvering av SP utan att tillåta access till underliggande tabeller.

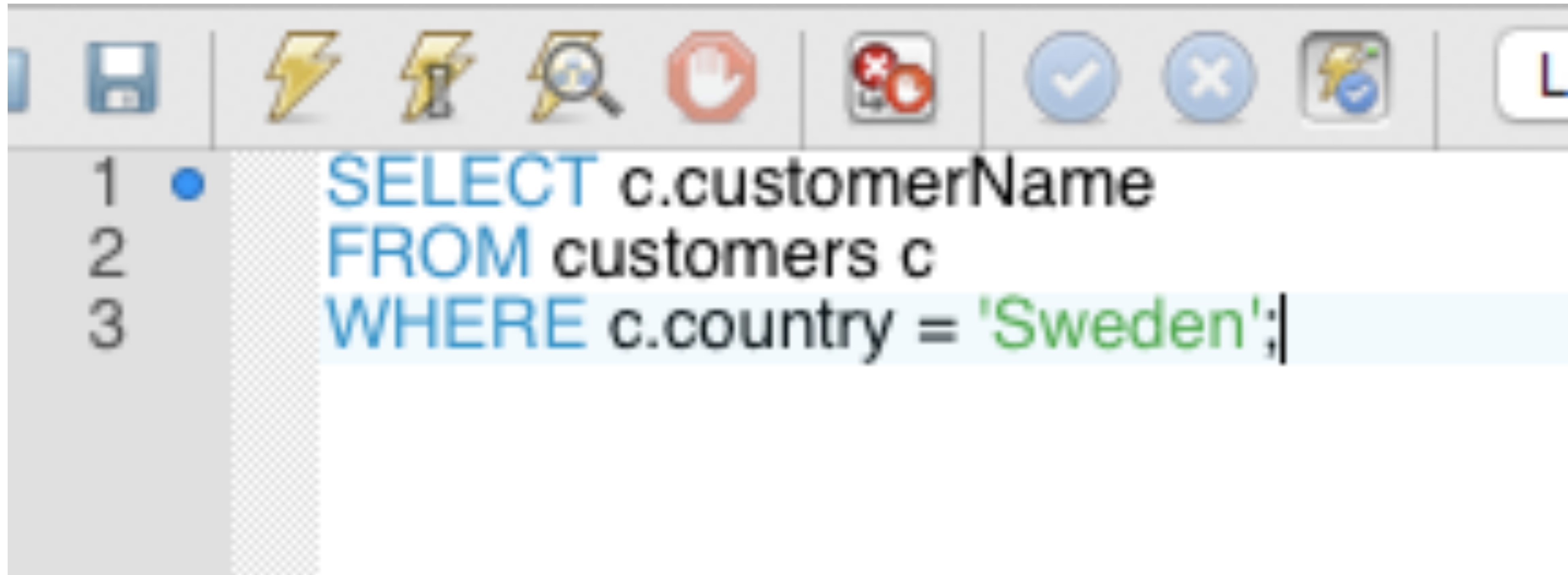
Stored procedure

Bättre prestanda

- En klient som kör flera SQL-frågor skapar extra nätverkstrafik och mer arbete för servern att tolka, validera och optimera varje sats.

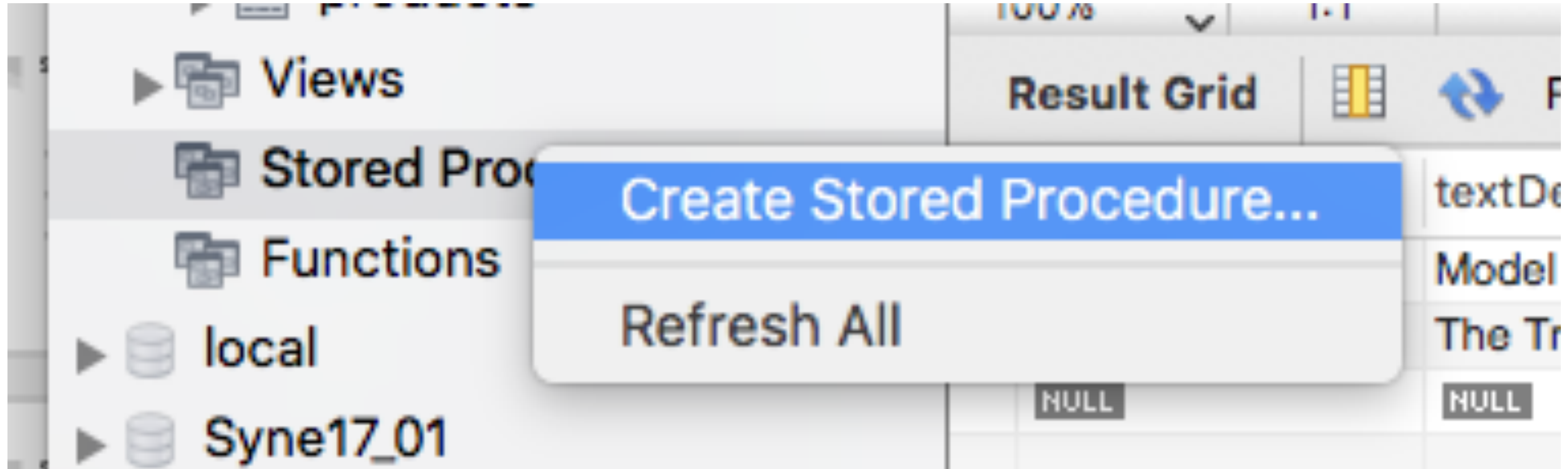
Stored procedure

Skapa och köra



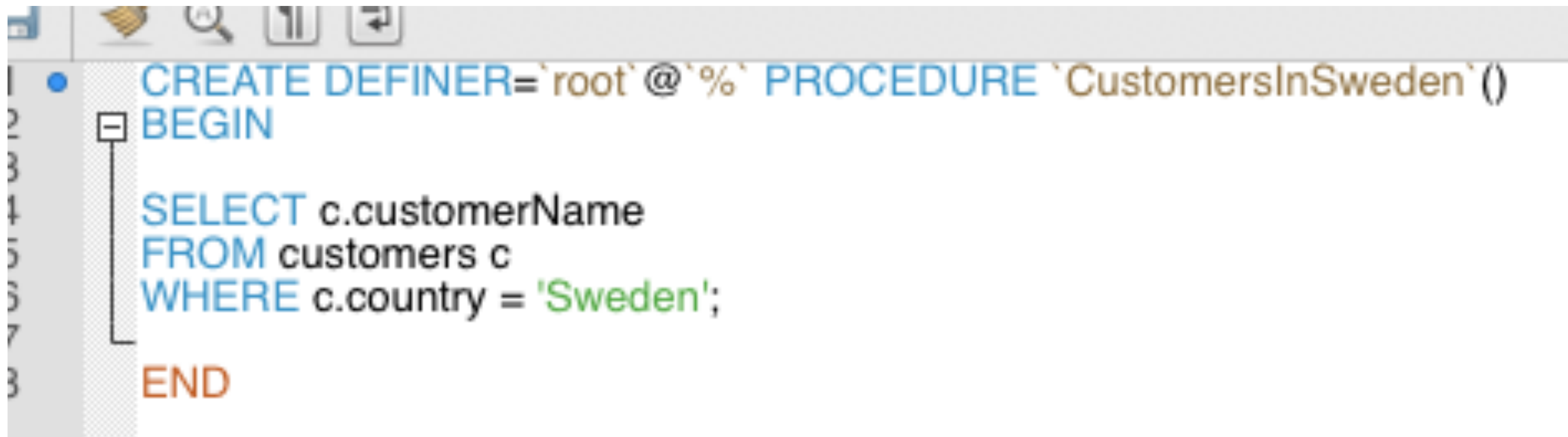
Stored procedure

Skapa och köra



Stored procedure

Skapa och köra



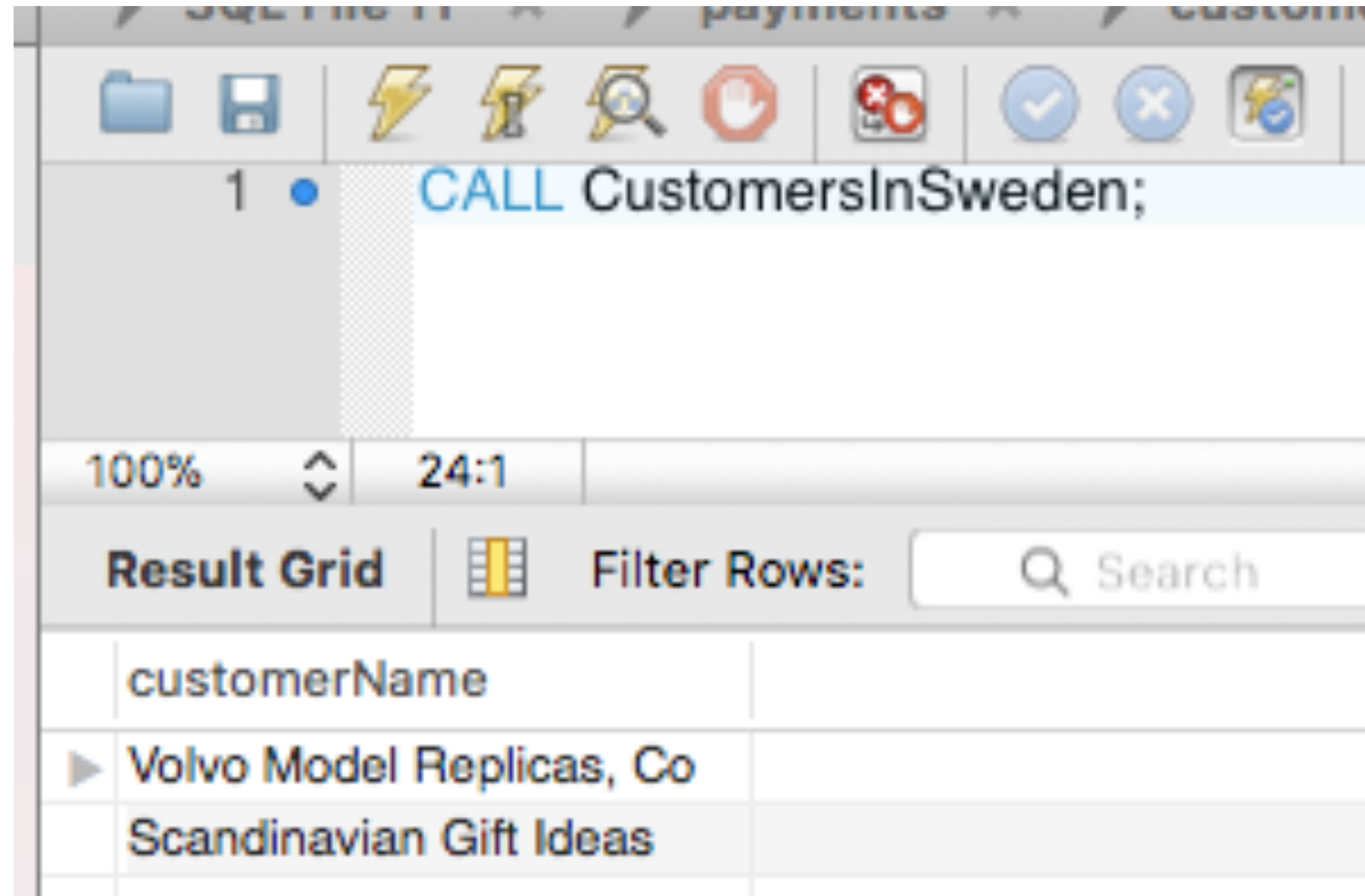
A screenshot of a SQL IDE interface. The top toolbar contains icons for file operations, execution, search, and undo/redo. The main editor area shows a SQL script for creating a stored procedure. The script is as follows:

```
1 CREATE DEFINER='root' @'%' PROCEDURE `CustomersInSweden` ()  
2 BEGIN  
3  
4 SELECT c.customerName  
5 FROM customers c  
6 WHERE c.country = 'Sweden';  
7  
8 END
```

The script is color-coded: 'CREATE DEFINER' is blue, 'root' is brown, '@' is black, '%' is black, 'PROCEDURE' is blue, 'CustomersInSweden' is brown, and '()' is black. 'BEGIN' is blue. 'SELECT' is blue, 'c.customerName' is black. 'FROM' is blue, 'customers c' is black. 'WHERE' is blue, 'c.country' is black, '=' is black, and 'Sweden' is green. 'END' is brown. Line numbers 1 through 8 are visible on the left margin.

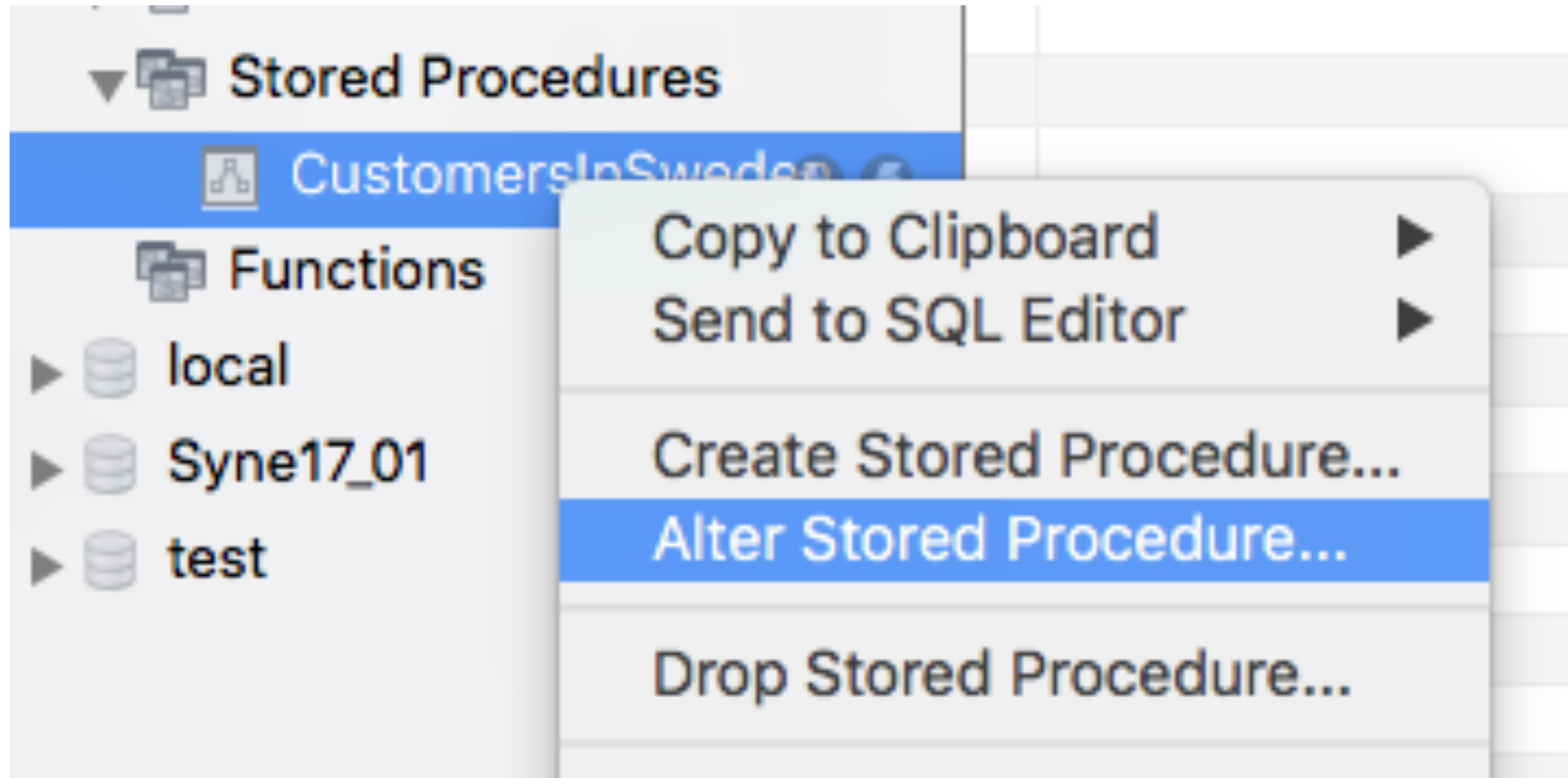
Stored procedure

Skapa och köra



Stored procedure

Ändra



Stored procedure

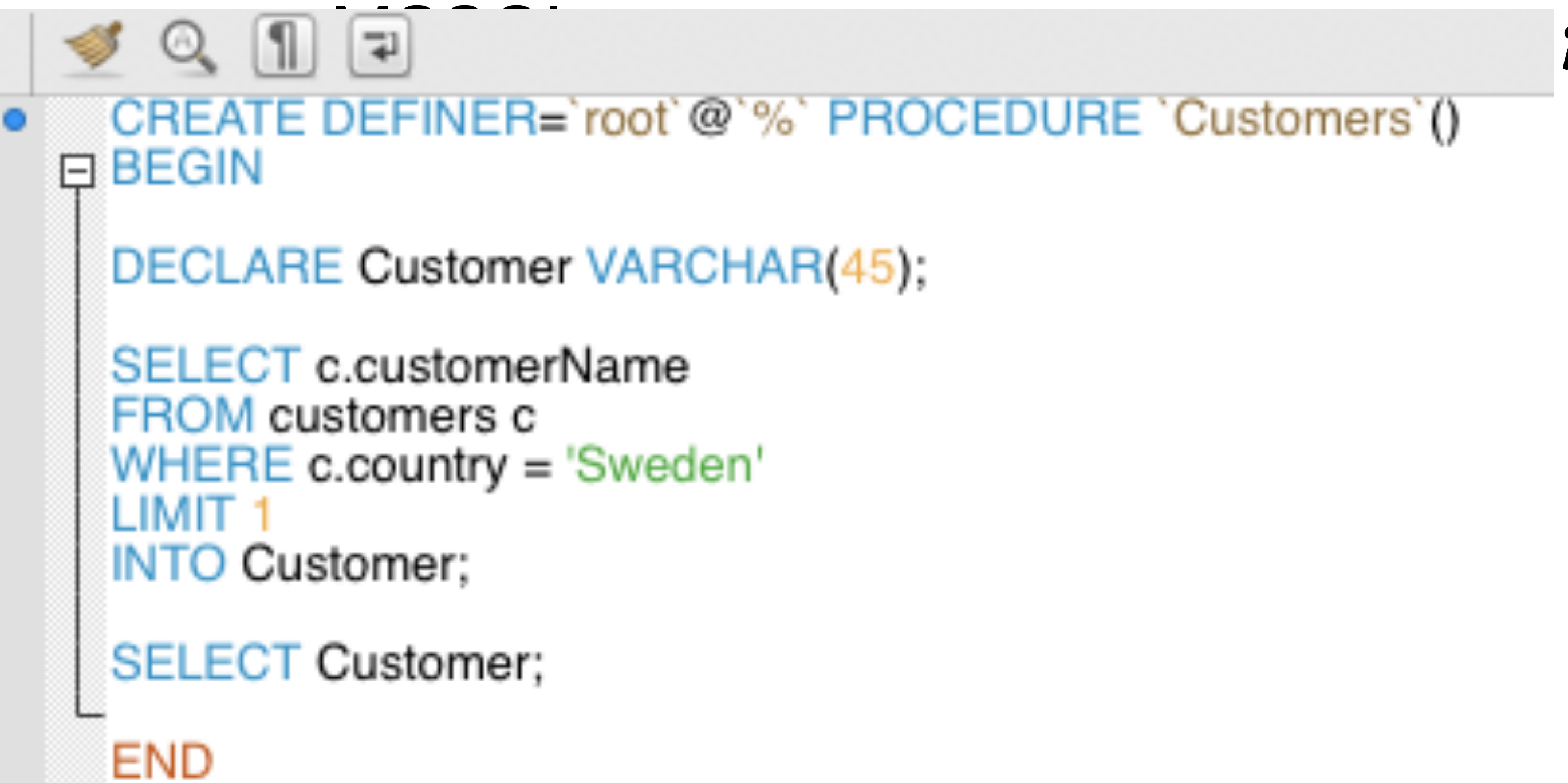
Lokala variabler

- För mer avancerade SP krävs variabler för att lagra delresultat.
 - `DECLARE variabel TYP;`
 - MSSQL: `DECLARE @variabel TYP;`

Stored procedure

Lokala variabler


- För mer avancerade SP krävs variabler för att lagra delresultat.
- `DECLARE variabel TYP;`



```
CREATE DEFINER='root'@'%' PROCEDURE `Customers`()  
BEGIN  
    DECLARE Customer VARCHAR(45);  
    SELECT c.customerName  
    FROM customers c  
    WHERE c.country = 'Sweden'  
    LIMIT 1  
    INTO Customer;  
    SELECT Customer;  
END;
```

Stored procedure

Lokala variabler

3 4	●	CALL Customers;
100%	↕	1:3
Result Grid		Filter Rows: <input type="text"/>
	Customer	
▶	Volvo Model Replicas, Co	

Stored procedure

Tilldela variabler

- `SELECT Customer = c.customerName
FROM customers c
WHERE country = 'Sweden';`
- `SET Customer = 'Volvo';`
- `SELECT Customer = 'Volvo';`

Globala variabler

4
5 ●
6

SHOW GLOBAL VARIABLES;

100%
1:5

Result Grid

Filter Rows:

Search

Variable_name	Value
auto_increment_offset	1
autocommit	ON
automatic_sp_privileges	ON
avoid_temporal_upgrade	OFF
back_log	80
basedir	/usr/
big_tables	OFF

In-parametrar

```
CREATE DEFINER='root'@'%' PROCEDURE `CustomersInCountry`(  
  p_country VARCHAR(45)  
)  
BEGIN  
  SELECT c.customerName  
  FROM customers c  
  WHERE c.country = p_country;  
END
```

```
CALL CustomersInCountry('Italy');
```

customerName	
▶ Amica Models & Co.	
Rovelli Gifts	
L'ordine Souvenirs	
Frau da Collezione	

In-parametrar

Default-värden

- MSSQL har stöd för default-värden:
- ```
CREATE PROCEDURE ProcName (
 @ParamName datatype = default
)
```
- MySQL har ännu inget stöd för default-värde.

# Parametrar

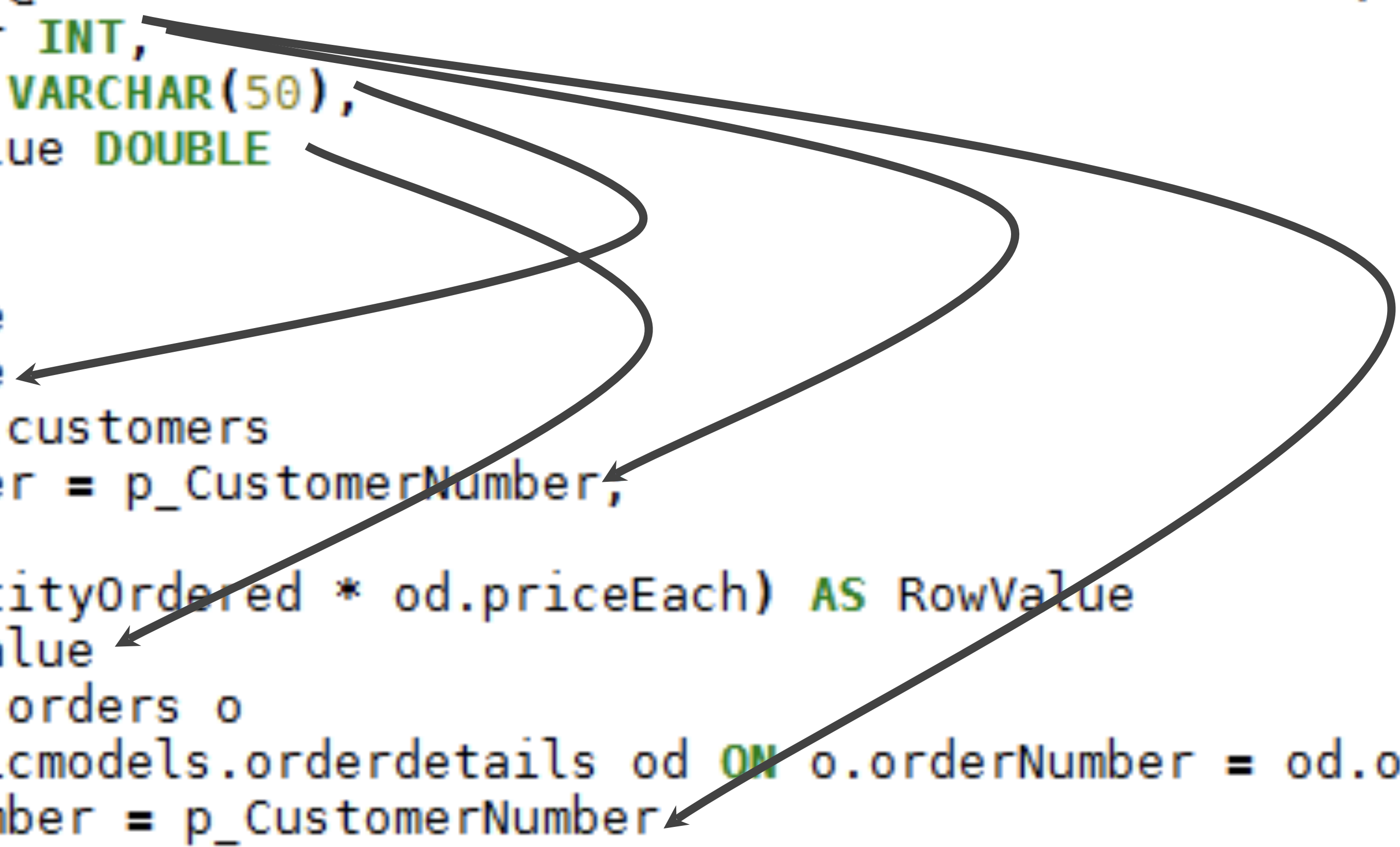
## In & Out

- Parametrar kan vara av typ IN, OUT eller INOUT.
  - IN är ett värde du skickar som parameter till SP:n.
  - OUT är ett värde du får tillbaka från SP:n.
  - INOUT kan göra både och.

# Parametrar

## In & Out

- ```
CREATE DEFINER=`root`@`localhost` PROCEDURE `GetOrdersPerCustomer` (  
  IN p_CustomerNumber INT,  
  OUT p_CustomerName VARCHAR(50),  
  OUT p_TotalOrderValue DOUBLE  
)  
BEGIN  
  SELECT customerName  
  INTO p_CustomerName  
  FROM classicmodels.customers  
  WHERE customerNumber = p_CustomerNumber;  
  
  SELECT SUM(od.quantityOrdered * od.priceEach) AS RowValue  
  INTO p_TotalOrderValue  
  FROM classicmodels.orders o  
    INNER JOIN classicmodels.orderdetails od ON o.orderNumber = od.orderNumber  
  WHERE o.customerNumber = p_CustomerNumber  
  GROUP BY o.customerNumber;  
END
```






Hur man kör

- Variabel som innehåller returvärde.
- Måste starta med @. Förutom det gäller vanliga namngivningsregler.
- Använd `SELECT` för returvärden.

Hur man kör

```
8
9 ● CALL GetOrdersPerCustomer(128, @CustomerName, @TotalValue);
10
11 ● SELECT @CustomerName, @TotalValue;
```

100% 35:11

Result Grid   Filter Rows: Export: 

	@CustomerName	@TotalValue
▶	Blauer See Auto, Co.	75937.76

DECLARE

- Du kan använda DECLARE för att deklarerera en lokal variabel i en SP.
- Måste deklareras efter BEGIN men innan något annat statement.

```
CREATE DEFINER='root'@'%' PROCEDURE `GetProducts`(  
  OUT p_ProductLine VARCHAR(45)  
)  
BEGIN  
  DECLARE sProductLine VARCHAR(50);  
  SET sProductLine = 'Motorcycles';  
  SET p_ProductLine = sProductLine;  
END
```


Villkorade statements

```
1 IF expression THEN commands  
2   END IF;
```

```
1 IF expression THEN commands  
2   ELSE commands  
3   END IF;
```

```
1 IF expression THEN commands  
2   ELSEIF expression THEN commands  
3   ELSE commands  
4   END IF;
```



IF

```
CREATE DEFINER='root'@'%' PROCEDURE `CheckStock`(  
  IN p_ProductCode VARCHAR (15),  
  OUT p_Status VARCHAR (45)  
)  
BEGIN  
  
  DECLARE nQuantity INT;  
  
  SELECT quantityInStock  
  INTO nQuantity  
  FROM products  
  WHERE productCode = p_ProductCode;  
  
  IF nQuantity < 100 THEN SET p_Status = "We're running low!";  
  ELSE SET p_Status = "Plenty in stock!";  
  END IF;  
  
END
```

IF

```
12
13 ● CALL CheckStock ('S10_1678', @Message);
14 ● CALL CheckStock ('S12_1099', @Message2);
15
16 ● SELECT @Message, @Message2;
```

00% 1:16

Result Grid   Filter Rows: **E**

@Message	@Message2
Plenty in stock!	We're running low!

Villkorade statements

```
1 CASE
2   WHEN expression THEN commands
3   ...
4   WHEN expression THEN commands
5   ELSE commands
6 END CASE;
```

CASE statements

```
CREATE DEFINER='root'@'%' PROCEDURE `CheckStock2` (  
  IN p_ProductCode VARCHAR (15),  
  OUT p_Status VARCHAR (45)  
)  
BEGIN  
  
  DECLARE nQuantity INT;  
  
  SELECT quantityInStock  
  INTO nQuantity  
  FROM products  
  WHERE productCode = p_ProductCode;  
  
  CASE  
  WHEN nQuantity > 5000 THEN SET p_Status = "We have a lot of these!";  
  WHEN nQuantity > 500 THEN SET p_Status = "We have a fair number of these!";  
  ELSE SET p_Status = "We're running low on these.";  
  END CASE;  
  
END
```


CASE statements

productCode	quantityInStock
S10_1678	7933
S10_4757	3252
S12_1099	68

```
1 CALL CheckStock ('S10_1678', @Message, @NoInStock1);
2 CALL CheckStock ('S12_1099', @AnotherMessage, @NoInStock2);
3 CALL CheckStock ('S10_4757', @ThirdMessage, @NoInStock3);
4
5 SELECT @Message, @NoInStock1, @AnotherMessage, @NoInStock2, @ThirdMessage, @NoInStock3;
6
```

Result (1) ×						
@Message	@NoInStock1	@AnotherMessage	@NoInStock2	@ThirdMessage	@NoInStock3	
We have a lot of these!	7933	We're starting to run low on these.	68	We have a fair number of these!	3252	

LOOP - WHILE

```
01 DELIMITER $$
02 DROP PROCEDURE IF EXISTS WhileLoopProc$$
03 CREATE PROCEDURE WhileLoopProc()
04     BEGIN
05         DECLARE x INT;
06         DECLARE str VARCHAR(255);
07         SET x = 1;
08         SET str = '';
09         WHILE x <= 5 DO
10             SET str = CONCAT(str,x,',');
11             SET x = x + 1;
12         END WHILE;
13         SELECT str;
14     END$$
15 DELIMITER ;
```

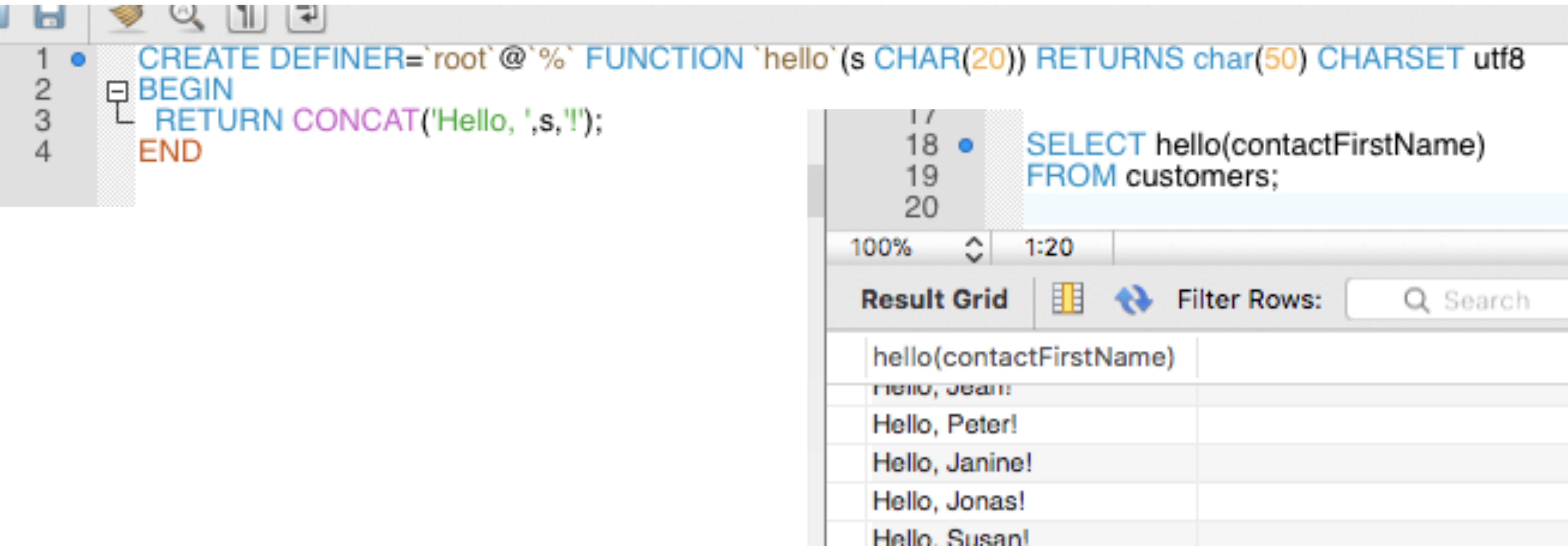
LOOP - REPEAT

```
01 DELIMITER $$
02 DROP PROCEDURE IF EXISTS RepeatLoopProc$$
03 CREATE PROCEDURE RepeatLoopProc()
04     BEGIN
05         DECLARE x INT;
06         DECLARE str VARCHAR(255);
07         SET x = 1;
08         SET str = '';
09         REPEAT
10             SET str = CONCAT(str,x,',');
11             SET x = x + 1;
12         UNTIL x > 5
13         END REPEAT;
14         SELECT str;
15     END$$
16 DELIMITER ;
```


LOOP - LEAVE & ITERATE

```
01 DELIMITER $$
02 DROP PROCEDURE IF EXISTS LOOPLoopProc$$
03 CREATE PROCEDURE LOOPLoopProc()
04     BEGIN
05         DECLARE x INT;
06         DECLARE str VARCHAR(255);
07         SET x = 1;
08         SET str = '';
09         loop_label: LOOP
10             IF x > 10 THEN
11                 LEAVE loop_label;
12             END IF;
13             SET x = x + 1;
14             IF (x mod 2) THEN
15                 ITERATE loop_label;
16             ELSE
17                 SET str = CONCAT(str,x,',');
18             END IF;
19
20         END LOOP;
21         SELECT str;
22     END$$
23 DELIMITER ;
```

User Defined Functions



The screenshot displays a SQL IDE interface. On the left, a code editor shows the creation of a user-defined function named 'hello'. The function takes a string 's' of type CHAR(20) and returns a string of type char(50) using the utf8 character set. The function body concatenates the string 'Hello, ' with the input 's' followed by an exclamation mark. The code is as follows:

```
1 CREATE DEFINER='root'@'%' FUNCTION `hello` (s CHAR(20)) RETURNS char(50) CHARSET utf8
2 BEGIN
3   RETURN CONCAT('Hello, ',s,'!');
4 END
```

On the right, a query window shows the execution of the function. The query selects the result of the 'hello' function applied to the 'contactFirstName' column of the 'customers' table. The query is:

```
17
18 SELECT hello(contactFirstName)
19 FROM customers;
20
```

Below the query, the 'Result Grid' is visible, showing the output of the query. The first column is labeled 'hello(contactFirstName)' and contains the following values:

hello(contactFirstName)
Hello, Jean!
Hello, Peter!
Hello, Janine!
Hello, Jonas!
Hello, Susan!

CLI vs GUI

- Command Line Interface
- Graphical User Interface
- När är vilket att föredra?

Exportera/importera data

The screenshot shows the MySQL Data Export wizard interface. The left sidebar contains navigation options under 'MANAGEMENT', 'INSTANCE', and 'PERFORMANCE'. The main area is titled 'Data Export' and shows the 'Object Selection' tab. A table lists schemas to export, with 'classicmodels' selected. Another table lists schema objects to export, all of which are checked.

MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE

- Startup / Shutdown
- Server Logs
- Options File

PERFORMANCE

- Dashboard
- Performance Reports
- Performance Schema Setup

Data Export

Object Selection | Export Progress

Tables to Export

Export	Schema
<input type="checkbox"/>	Syne17_01
<input checked="" type="checkbox"/>	classicmodels
<input type="checkbox"/>	local
<input type="checkbox"/>	test

Export	Schema Objects
<input checked="" type="checkbox"/>	customers
<input checked="" type="checkbox"/>	employees
<input checked="" type="checkbox"/>	offices
<input checked="" type="checkbox"/>	orderdetails
<input checked="" type="checkbox"/>	orders
<input checked="" type="checkbox"/>	payments
<input checked="" type="checkbox"/>	productlines
<input checked="" type="checkbox"/>	products
<input checked="" type="checkbox"/>	vw_stockValue

Exportera/importera data

Refresh

9 table

Dump Structure and Data

Select Views

Select Tables

Unselect All

Objects to Export

☐ Dump Stored Procedures and Functions

☐ Dump Events

☐ Dump Triggers

Export Options

☐ Export to Dump Project Folder

/Users/micke/dumps/Dump20180326

...

Each table will be exported into a separate file. This allows a selective restore, but may be slower.

☒ Export to Self-Contained File

/Users/micke/dumps/Dump20180326-1.sql

...

All selected database objects will be exported into a single, self-contained file.

☐ Create Dump in a Single Transaction (self-contained file only)

☐ Include Create Schema

Export Completed

Start Export

Exportera/importera data

- Self-contained ger en sql-fil likt `classic_models.sql`.

Sammanfattning

- Aggregerade funktioner
 - Funktioner som gör beräkningar på grupper av data
- Funktioner
 - Några av de inbyggda funktionerna i MySQL
- Sub queries
 - Frågor med delfrågor eller underfrågor
- Skript
 - Automatiserade frågor
- Stored procedures
 - Kompilerade funktioner
- User Defined Functions
 - Egna funktioner
- Exportera / importera