



Backendprogrammering 1

# RELATIONSDBASER

Utbildare: Mikael Olsson

[mikael.olsson@emmio.se](mailto:mikael.olsson@emmio.se)

076-174 90 43

# NACKADEMIN



# 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**

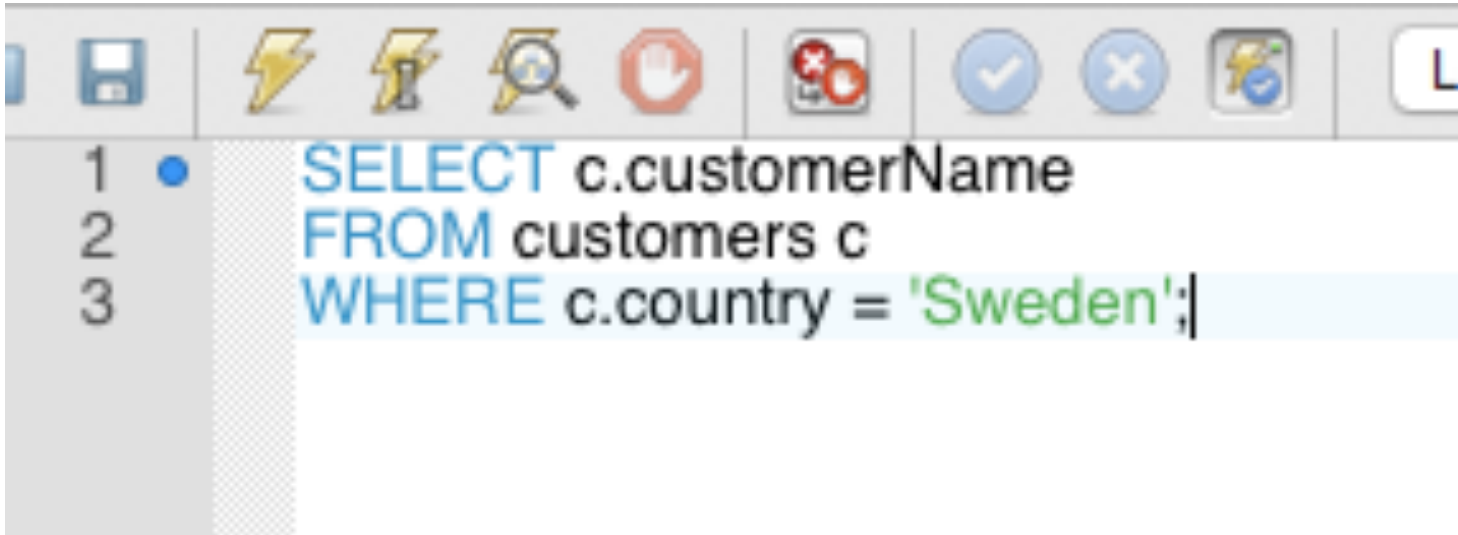
### **- bättre prestanda**

- 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

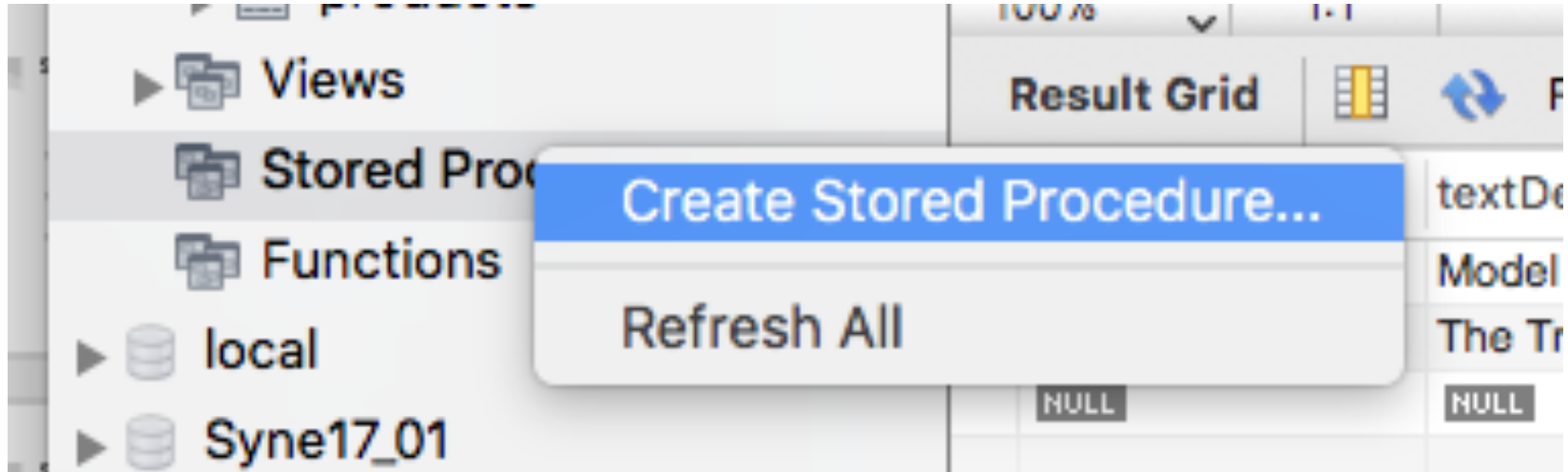


The image shows a screenshot of a SQL IDE interface. At the top is a toolbar with various icons: a save icon, a lightning bolt, a lightning bolt with a plug, a magnifying glass, a stop sign, a red circle with a white 'X', a blue circle with a white checkmark, a blue circle with a white 'X', a lightning bolt with a checkmark, and a button labeled 'L'. Below the toolbar is a code editor with a line number margin on the left showing lines 1, 2, and 3. The code being edited is a SQL query: `SELECT c.customerName` on line 1, `FROM customers c` on line 2, and `WHERE c.country = 'Sweden';` on line 3. The word 'Sweden' is highlighted in green.

```
1 SELECT c.customerName
2 FROM customers c
3 WHERE c.country = 'Sweden';
```

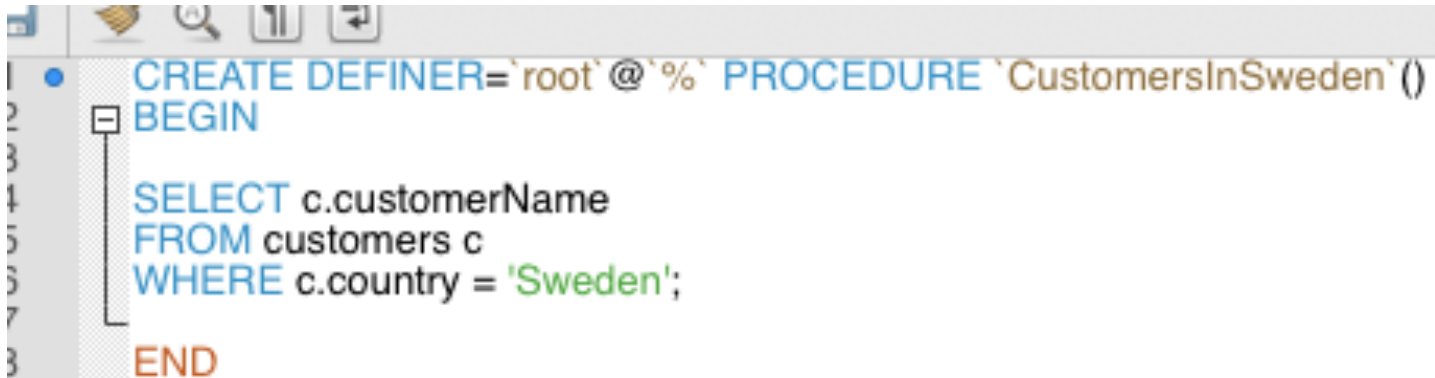
# Stored procedure

- skapa och köra



# Stored procedure

- skapa och köra

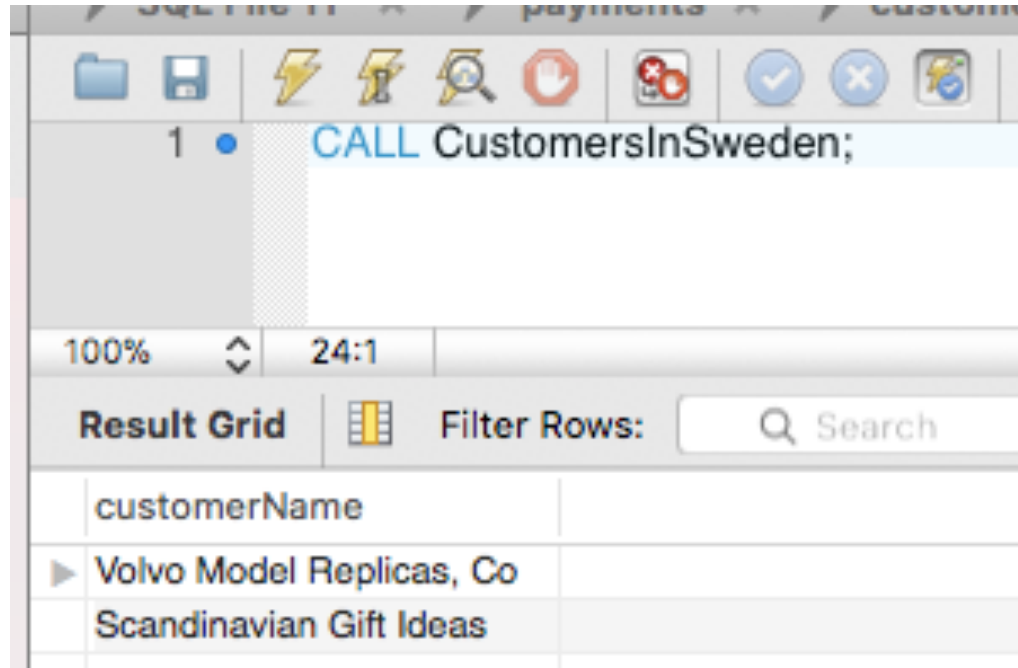


A screenshot of a SQL IDE interface. The top toolbar contains icons for file operations, execution, search, and navigation. The main editor area displays SQL code for creating a stored procedure. The code is as follows:

```
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
```

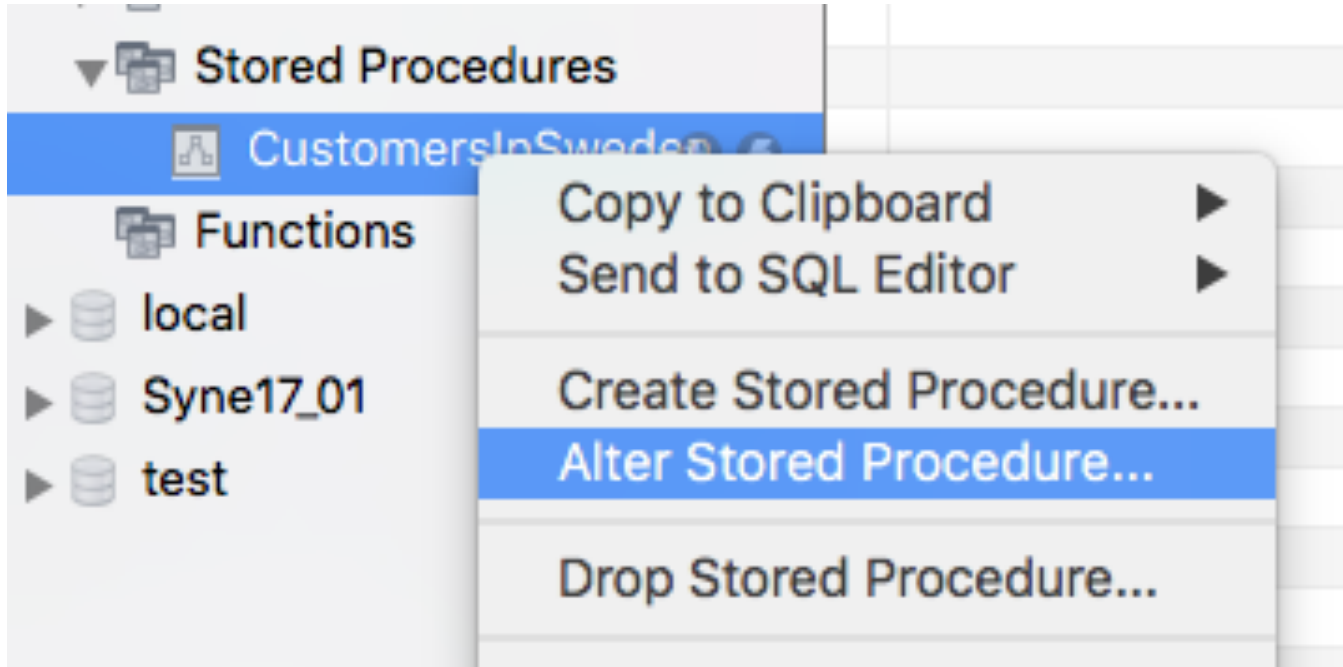
# Stored procedure

- skapa och köra



# Stored procedure

## - ändra





# Stored procedure

## - lokala variabler

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

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



## Stored procedure

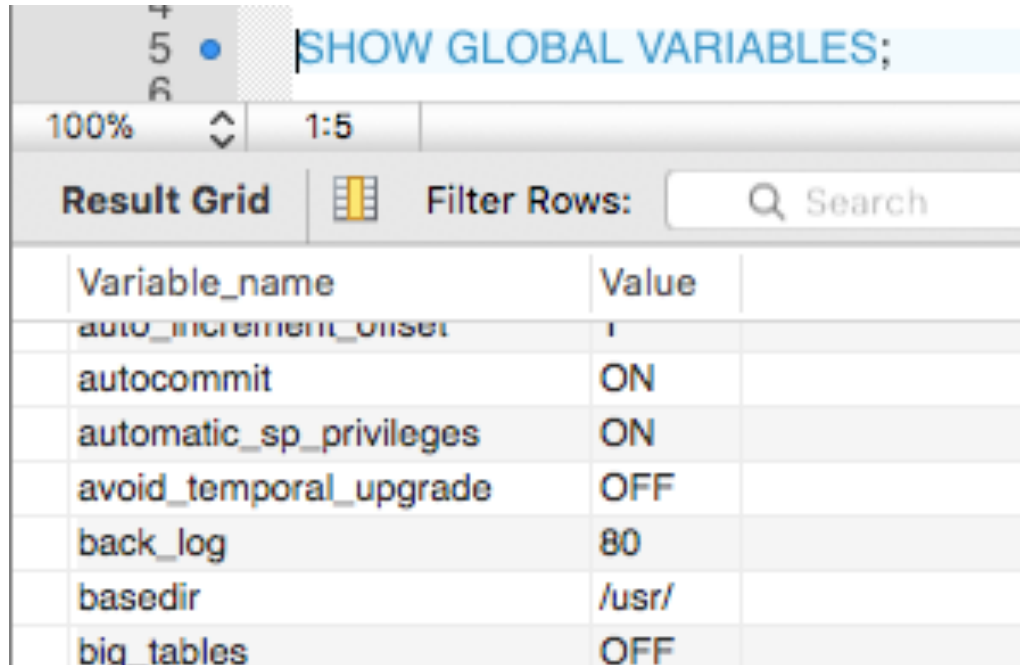
### - tilldela variabler

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





# Globala variabler



The screenshot shows a MySQL command window with the command `SHOW GLOBAL VARIABLES;` entered. Below the command, a 'Result Grid' displays the output. The grid has two columns: 'Variable\_name' and 'Value'. The first row is partially obscured by a scroll bar. The subsequent rows show various system variables and their current values.

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');
```

1:9



Filter Rows:



Search

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



# Parametrar

## - in & out

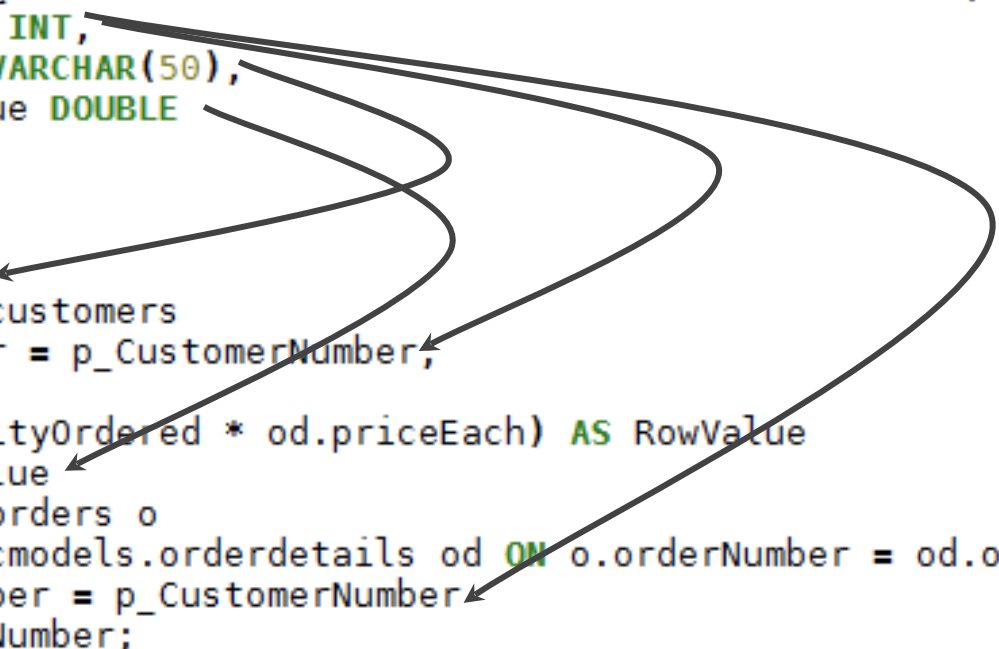
- Parametrar kan vara av typ `IN`, `OUT` eller `INOUT`.
  - `IN` är ett värde du skickar som parameter till SP:n.
  - `OUT` är ett värde du får tillbaka från SP:n.
  - `INOUT` kan göra både och.

# Parametrar

## - in & out

- IN är inte obligatoriskt för IN-parametrar, men det gör det tydligare.

```
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.

The screenshot shows a SQL IDE interface. The query editor displays two lines of SQL code:

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

Below the query editor, the status bar shows "100%" zoom and "35:11" cursor position. The "Result Grid" tab is active, displaying the results of the query in a table:

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

The interface also includes a "Filter Rows" section with a search box and an "Export" button with a save icon.



# DECLARE

- Du kan använda DECLARE för att deklarerera en lokal variabel i en SP.
- Måste deklareraras 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
```

view

Output

Snippets

Result (1) x

Fetch 1 records. Duration: 0.000 sec, fetched in: 0.000 sec

@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 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:

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

Below the query window, the 'Result Grid' is visible, showing the output of the function for each row in the 'customers' table. The first column is 'hello(contactFirstName)'.

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



# Exportera/importera data

The screenshot shows the MySQL Data Export wizard. The left sidebar contains navigation menus for Management, Instance, and Performance. The main window is titled 'Data Export' and shows the 'Object Selection' tab. It lists tables to export, with 'classicmodels' selected. A list of schema objects is also shown, all of which are checked for export.

**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

☐ Export to 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`.



# Exportera med CLI

- mysqldump
- <https://www.linode.com/docs/databases/mysql/use-mysqldump-to-back-up-mysql-or-mariadb/>



# CLI vs GUI

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



# CRON

- Schemalagd skriptkörning
- Cron is driven by a crontab (cron table) file, a configuration file that specifies shell commands to run periodically on a given schedule.
- Vi kan t ex använda detta för att sätta upp automatisk backup

Each line of a crontab file represents a job, and looks like this:

```
# |----- minute (0 - 59)
# |----- hour (0 - 23)
# |----- day of the month (1 - 31)
# |----- month (1 - 12)
# |----- day of the week (0 - 6) (Sunday to Saturday;
# |                                     7 is also Sunday on some systems)
#
# * * * * * command to execute
```

## Ex Local Host mysql Backup:

```
0 1 * * * /usr/local/mysql/bin/mysqldump -uroot -ppassword --opt database > /path/to/directory/filename.sql
```

(There is no space between the -p and password or -u and username - replace root with a correct database username.)