

Stored Procedures

Thomas Schwarz, SJ

Stored Routines

- Stored Routine:
 - SQL statement or set of SQL statements that can be stored in the database server
 - Can be a function
 - Can be a stored procedure

MySQL Stored Procedures

- Delimiters
 - Semicolon acts as a delimiter in SQL and Procedures
 - Need to change delimiter
 - Can be set to anything
 - E.g. double dollar sign
 - Afterwards reset it

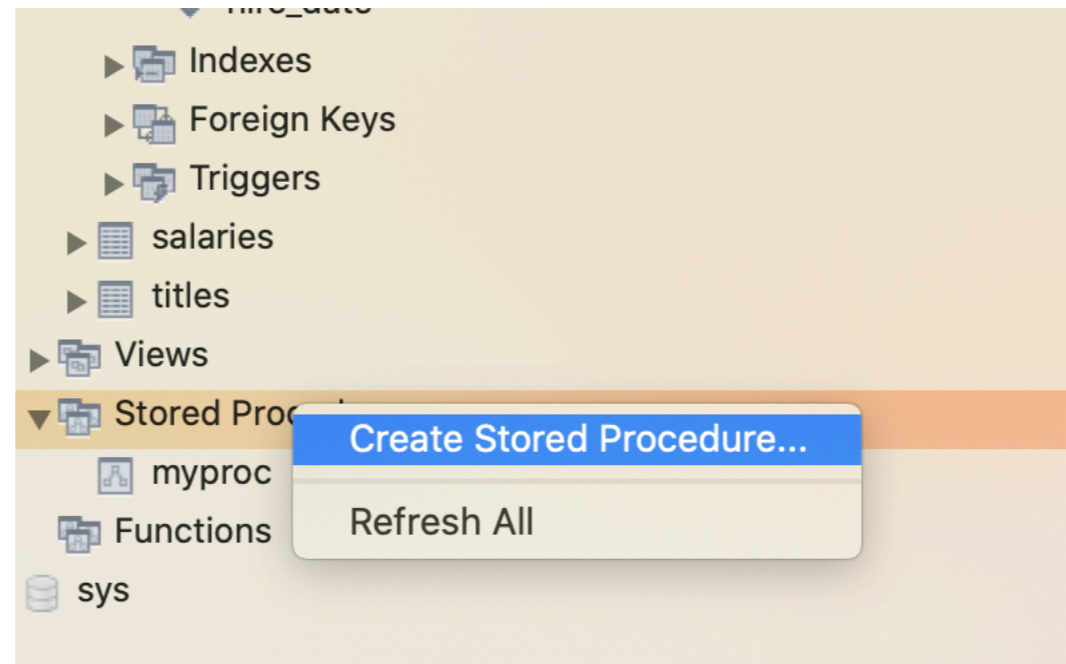
```
DELIMITER $$
```

```
...
```

```
DELIMITER ;
```

MySQL Stored Procedures

- We can generate stored procedures from within MySQL workbench
- Click on Stored Procedures at the end of the schema and select create stored procedure



MySQL Stored Procedures

- We can generate stored procedure using SQL

```
use employees;
```

```
DELIMITER $$
```

```
CREATE PROCEDURE myproc ()
```

```
    BEGIN
```

```
        SELECT *
```

```
            FROM employees
```

```
            WHERE first_name LIKE 'th%';
```

```
    END
```

```
DELIMITER ;
```

Keywords are CREATE
PROCEDURE

MySQL Stored Procedures

- We can generate stored procedure using SQL

```
use employees;
```

```
DELIMITER $$
```

```
CREATE PROCEDURE myproc ()
```

```
    BEGIN
```

```
        SELECT *
```

```
            FROM employees
```

```
            WHERE first_name LIKE 'th%';
```

```
    END
```

```
DELIMITER ;
```



Need to give it a name

MySQL Stored Procedures

- We can generate stored procedure using SQL

```
use employees;
```

```
DELIMITER $$
```

```
CREATE PROCEDURE myproc ()
```

```
    BEGIN
```

```
        SELECT *
```

```
            FROM employees
```

```
            WHERE first_name LIKE 'th%';
```

```
    END
```

```
DELIMITER ;
```



Can have arguments

MySQL Stored Procedures

- We can generate stored procedure using SQL

```
use employees;
```

```
DELIMITER $$
```

```
CREATE PROCEDURE myproc()
```

```
    BEGIN
```

```
        SELECT *
```

```
            FROM employees
```

```
            WHERE first_name LIKE 'th%';
```

```
    END
```

```
DELIMITER ;
```

**BEGIN END encapsulate
a SQL query terminated
with ;**

MySQL Stored Procedures

- We can generate stored procedure using SQL

```
use employees;  
  
DELIMITER $$  
CREATE PROCEDURE myproc()  
BEGIN  
    SELECT *  
    FROM employees  
    WHERE first_name LIKE 'th%';  
END  
DELIMITER ;
```

Don't forget to change
the delimiter back to ;

MySQL Stored Procedures

- Another procedure example:

```
CREATE PROCEDURE all_payments()  
  BEGIN  
    SELECT cus.customerName,  
           cus.customerNumber,  
           pay.checkNumber,  
           pay.paymentDate,  
           amount  
    FROM customers cus JOIN payments pay  
    USING (customerNumber);  
  END
```

MySQL Stored Procedures

- We can call a stored procedure
 - From within the workbench, by clicking on it
 - Using CALL procedureName()

```
4
5 CREATE PROCEDURE myproc()
6 BEGIN
7     SELECT *
8     FROM employee
9     WHERE first_name LIKE 'th%';
10 END
11
12 DELIMITER ;
13
14 CALL myproc();
15
```

100% 15:14

Result Grid

Filter Rows:

Search

Export:

emp_no	birth_date	first_name	last_name	gender	hire_date
11407	1962-01-11	Thanasis	Thebaut	M	1991-05-06
11422	1956-06-03	Thanasis	Ghalwash	F	1990-12-04
11825	1961-12-12	Theirry	Kuzuoka	F	1987-05-02
12158	1959-07-26	Theirry	Masada	M	1996-02-11
12403	1958-06-08	Thodoros	Samarati	F	1988-03-19
12565	1964-05-19	Theron	Mullainathan	M	1989-06-16
12569	1962-04-19	Thodoros	Lundstrom	F	1991-10-15
12594	1958-09-25	Thanasis	Ranst	F	1993-06-21
13212	1963-08-03	Thodoros	Boyle	F	1990-04-19
13356	1961-04-24	Thodoros	Lukaszewicz	M	1987-07-17
13463	1954-08-29	Theron	Berstel	F	1988-01-26
13870	1952-04-25	Theirry	Kirkerud	F	1989-01-09
14159	1953-11-12	Theron	Marrakchi	F	1994-06-22
14479	1964-08-22	Thodoros	Suessmith	M	1986-06-14
14631	1963-01-18	Theirry	Pulkowski	M	1994-06-10
14845	1962-08-10	Theron	Homond	F	1986-01-01
15000	1959-11-29	Thanasis	Bahi	F	1988-03-27

MySQL Stored Procedures

- Another procedure example:
 - Do not forget to reset the delimiter

```
CALL all_payments;
```

```
[ -> ;
```

customerName	customerNumber	checkNumber	paymentDate	amount
Atelier graphique	103	HQ336336	2004-10-19	6066.78
Atelier graphique	103	JM555205	2003-06-05	14571.44
Atelier graphique	103	OM314933	2004-12-18	1676.14
Signal Gift Stores	112	B0864823	2004-12-17	14191.12
Signal Gift Stores	112	HQ55022	2003-06-06	32641.98
Signal Gift Stores	112	ND748579	2004-08-20	33347.88
Australian Collectors, Co.	114	GG31455	2003-05-20	45864.03
Australian Collectors, Co.	114	MA765515	2004-12-15	82261.22
Australian Collectors, Co.	114	NP603840	2003-05-31	7565.08
Australian Collectors, Co.	114	NR27552	2004-03-10	44894.74
La Rochelle Gifts	119	DB933704	2004-11-14	19501.82
La Rochelle Gifts	119	LN373447	2004-08-08	47924.19
La Rochelle Gifts	119	NG94694	2005-02-22	49523.67
Baane Mini Imports	121	DB889831	2003-02-16	50218.95
Baane Mini Imports	121	FD317790	2003-10-28	1491.38
Baane Mini Imports	121	KI831359	2004-11-04	17876.32
Baane Mini Imports	121	MA302151	2004-11-28	34638.14
Mini Gifts Distributors Ltd.	124	AE215433	2005-03-05	101244.59
Mini Gifts Distributors Ltd.	124	BG255406	2004-08-28	85410.87
Mini Gifts Distributors Ltd.	124	CQ287967	2003-04-11	11044.30

MySQL Stored Procedures

- Procedure parameters have three types
 - IN — input
 - OUT — output
 - INOUT — both input and output
- Procedure parameters have type
 - Definition of parameter:

IN
OUT
INOUT

parameter name	type
-------------------	------

,

MySQL Stored Procedures

- Example: payments by one customer number

```
CREATE PROCEDURE payments_by(IN cnr INT)
  BEGIN
    SELECT cus.customerName,
           cus.customerNumber,
           pay.checkNumber,
           pay.paymentDate,
           amount
    FROM customers cus JOIN payments pay
    USING(customerNumber)
    WHERE customerNumber = cnr;
  END
```

MySQL Stored Procedures

```
[mysql> CALL payments_by(456)$$
```

customerName	customerNumber	checkNumber	paymentDate	amount
Microscale Inc.	456	GJ715659	2004-11-13	27550.51
Microscale Inc.	456	M0743231	2004-04-30	1679.92

```
2 rows in set (0.01 sec)
```

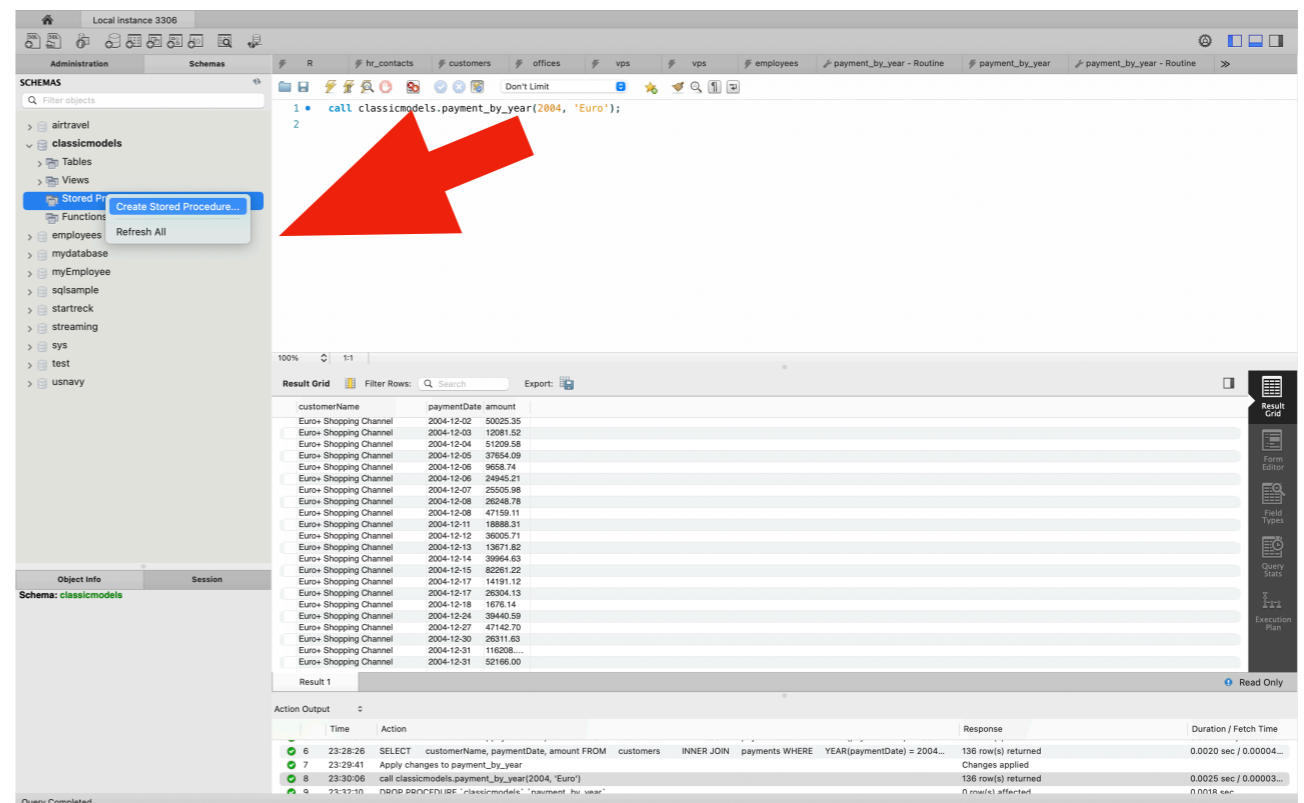
```
Query OK, 0 rows affected (0.01 sec)
```

MySQL Stored Procedures

- Find all payments by customer name and year
 - How to get the year?
 - Use YEAR(my_date) function
 - Gives an INT
 - How to get customer name?
 - Use Like and concatenate

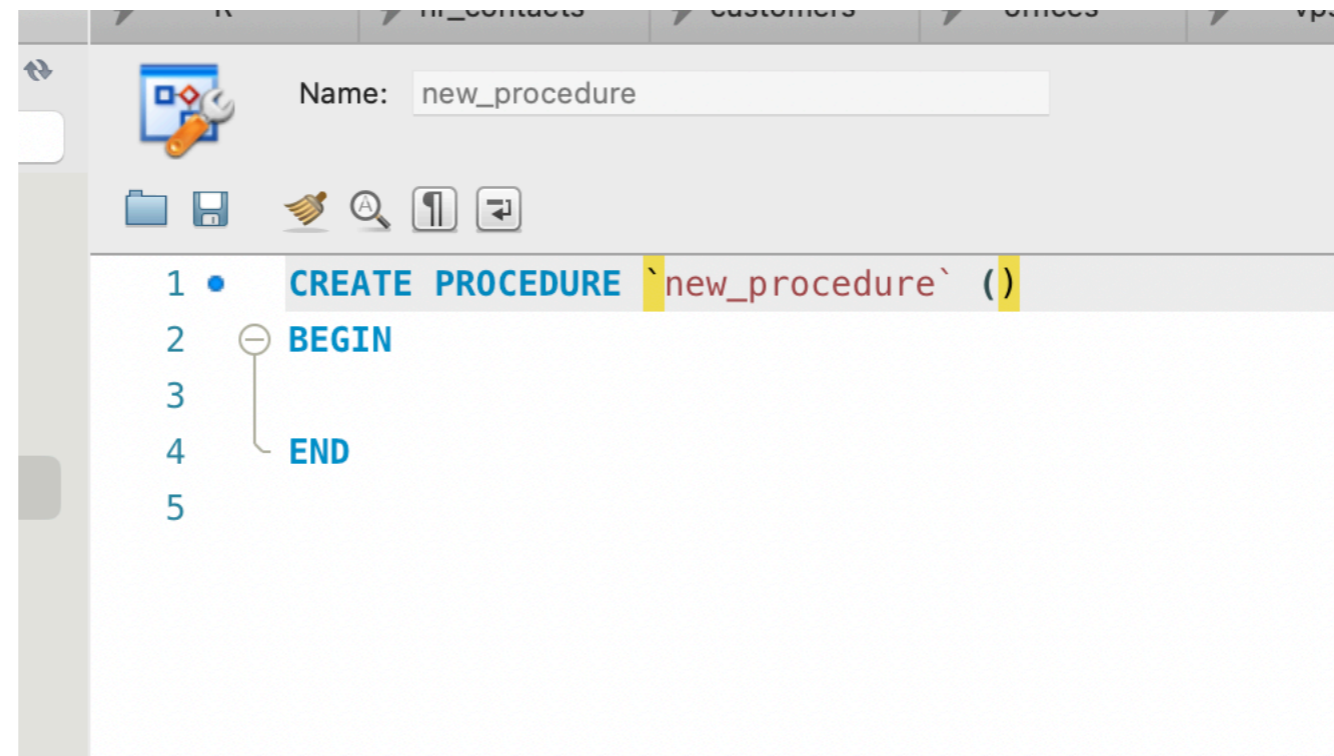
MySQL Stored Procedures

- It's easier in MySQLWorkbench:
 - In the schema panel on the left select Stored Procedures
 - Select Create



MySQL Stored Procedures

- A new query tab appears which you can edit



The screenshot shows a MySQL query editor interface. At the top, there is a tab labeled 'Name: new_procedure'. Below the tab, there is a toolbar with icons for file operations and editing. The main area contains the following SQL code:

```
1 CREATE PROCEDURE `new_procedure` ()  
2 BEGIN  
3  
4 END  
5
```

MySQL Stored Procedures

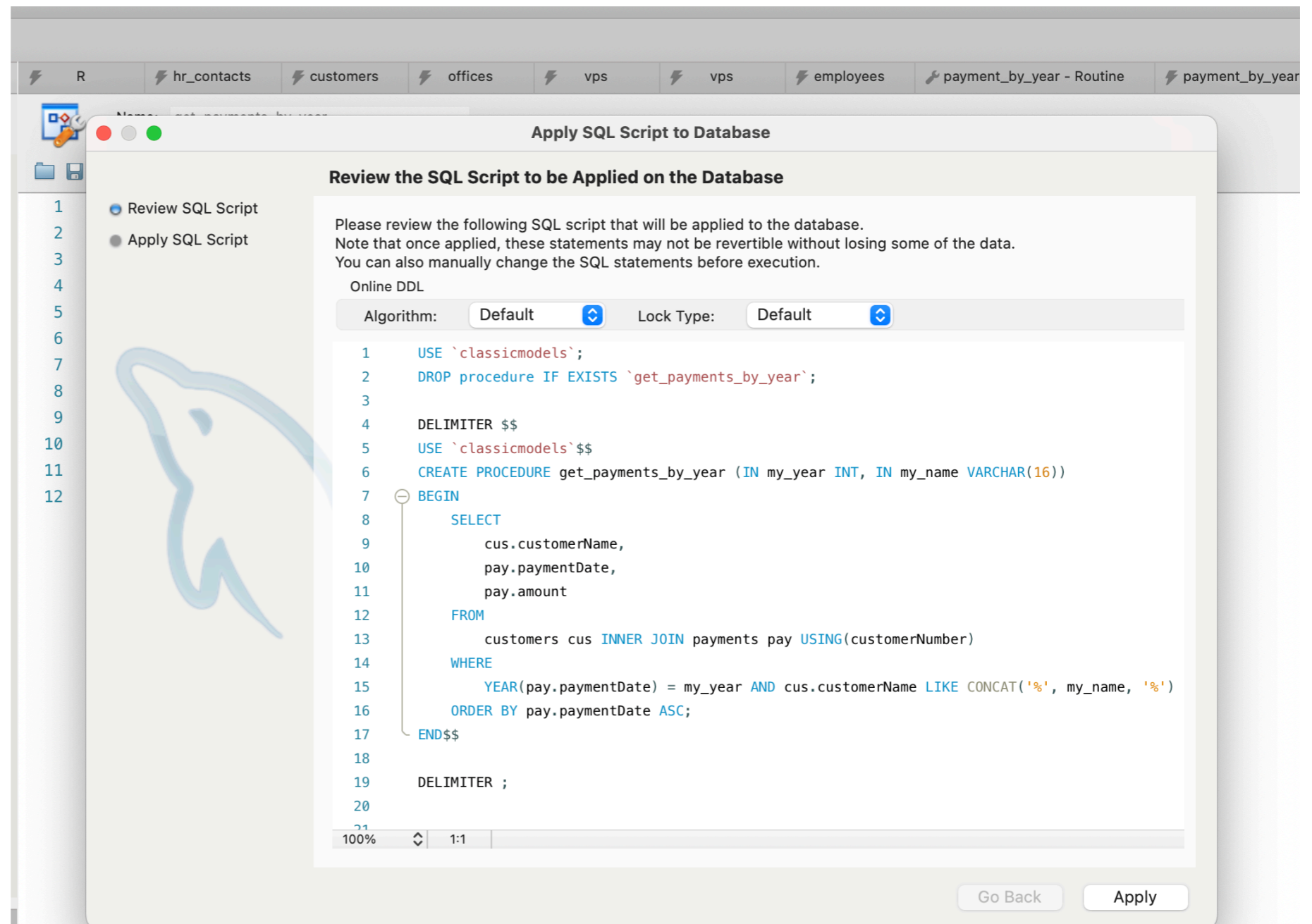
- We fill in the blank between Begin and END
- Press apply in the right bottom



```
CREATE DEFINER=`root`@`localhost` PROCEDURE `get_payments_by_year` (IN my_year INT, IN my_name VARCHAR(16))
BEGIN
  SELECT
    cus.customerName,
    pay.paymentDate,
    pay.amount
  FROM
    customers cus INNER JOIN payments pay USING(customerNumber)
  WHERE
    YEAR(pay.paymentDate) = my_year AND cus.customerName LIKE CONCAT('%', my_name, '%')
  ORDER BY pay.paymentDate ASC;
END
```

MySQL Stored Procedures

- After pressing apply, you get to review what you have written



MySQL Stored Procedures

- After you applied again, the procedure is stored
- You can execute it with a query

```
1 • CALL get_payments_by_year(2003, 'Euro');
```

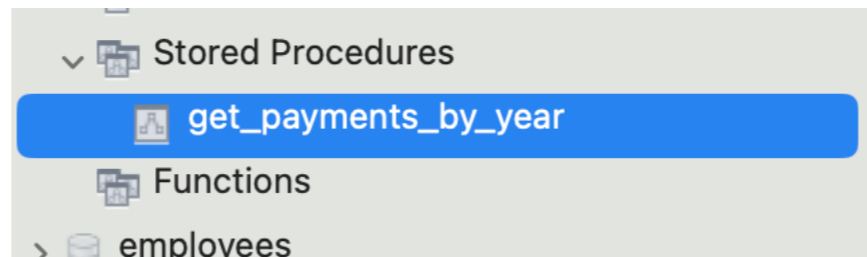
100% 40:1

Result Grid Filter Rows: Search Export:

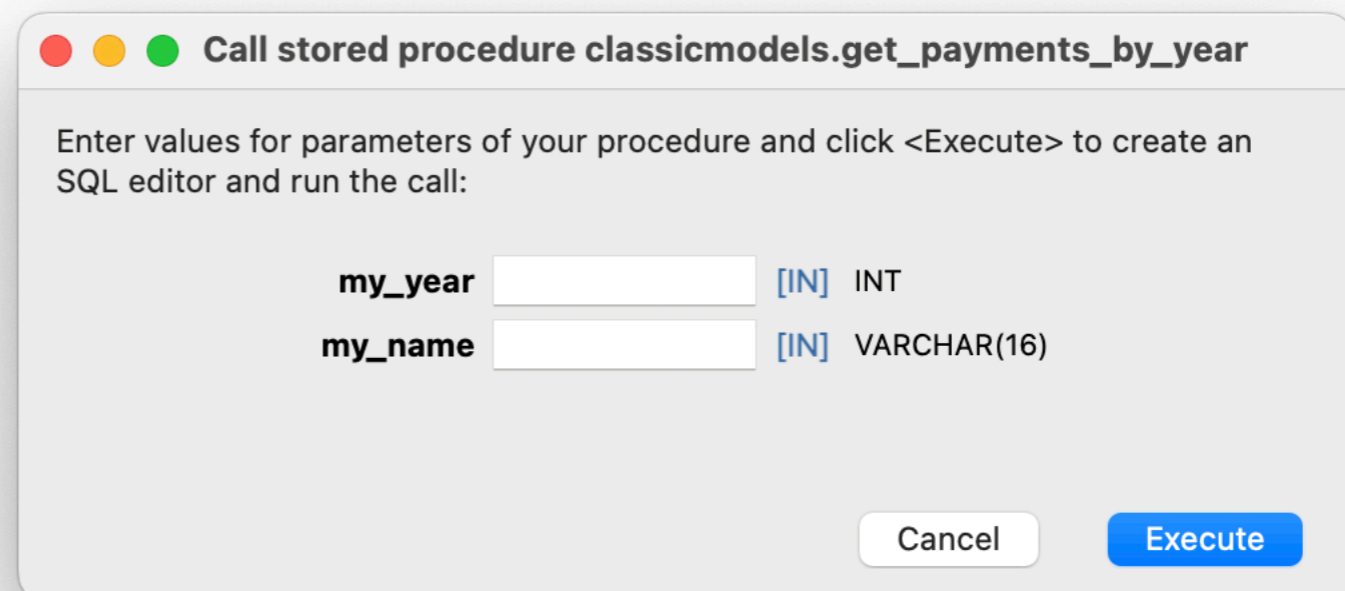
customerName	paymentDate	amount
Euro+ Shopping Channel	2003-02-25	40206.20
Euro+ Shopping Channel	2003-07-19	36251.03
Euro+ Shopping Channel	2003-10-26	49539.37
Euro+ Shopping Channel	2003-12-09	63843.55

MySQL Stored Procedures

- Or you can execute it from the left panel:
 - Select the stored procedure and press the flash symbol



- You get a window that asks you to insert the parameters



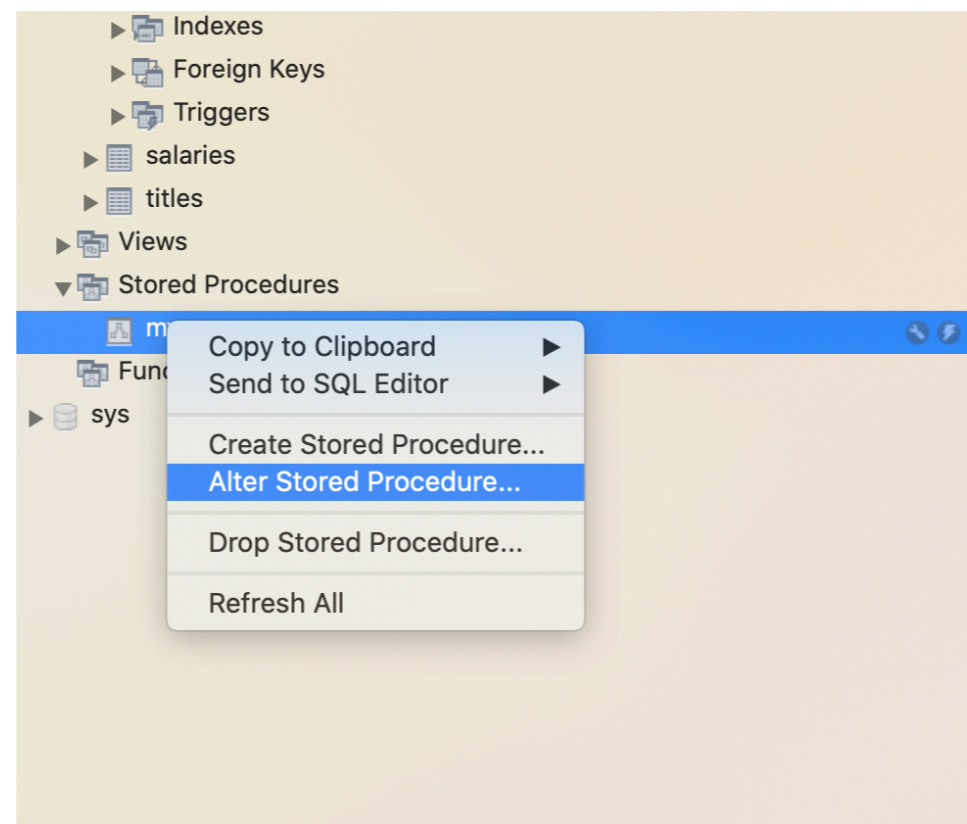
MySQL Stored Procedures

- And after filling them in, you press execute

Result Grid			
Filter Rows:		Search	Export:
customerName	paymentDate	amount	
▶ Euro+ Shopping Channel	2004-01-30	59830.55	
◀ Euro+ Shopping Channel	2004-05-17	26155.91	
◀ Euro+ Shopping Channel	2004-07-09	35420.74	
◀ Euro+ Shopping Channel	2004-08-16	20009.53	
◀ Euro+ Shopping Channel	2004-11-01	36140.38	
◀ Euro+ Shopping Channel	2004-12-31	116208.40	

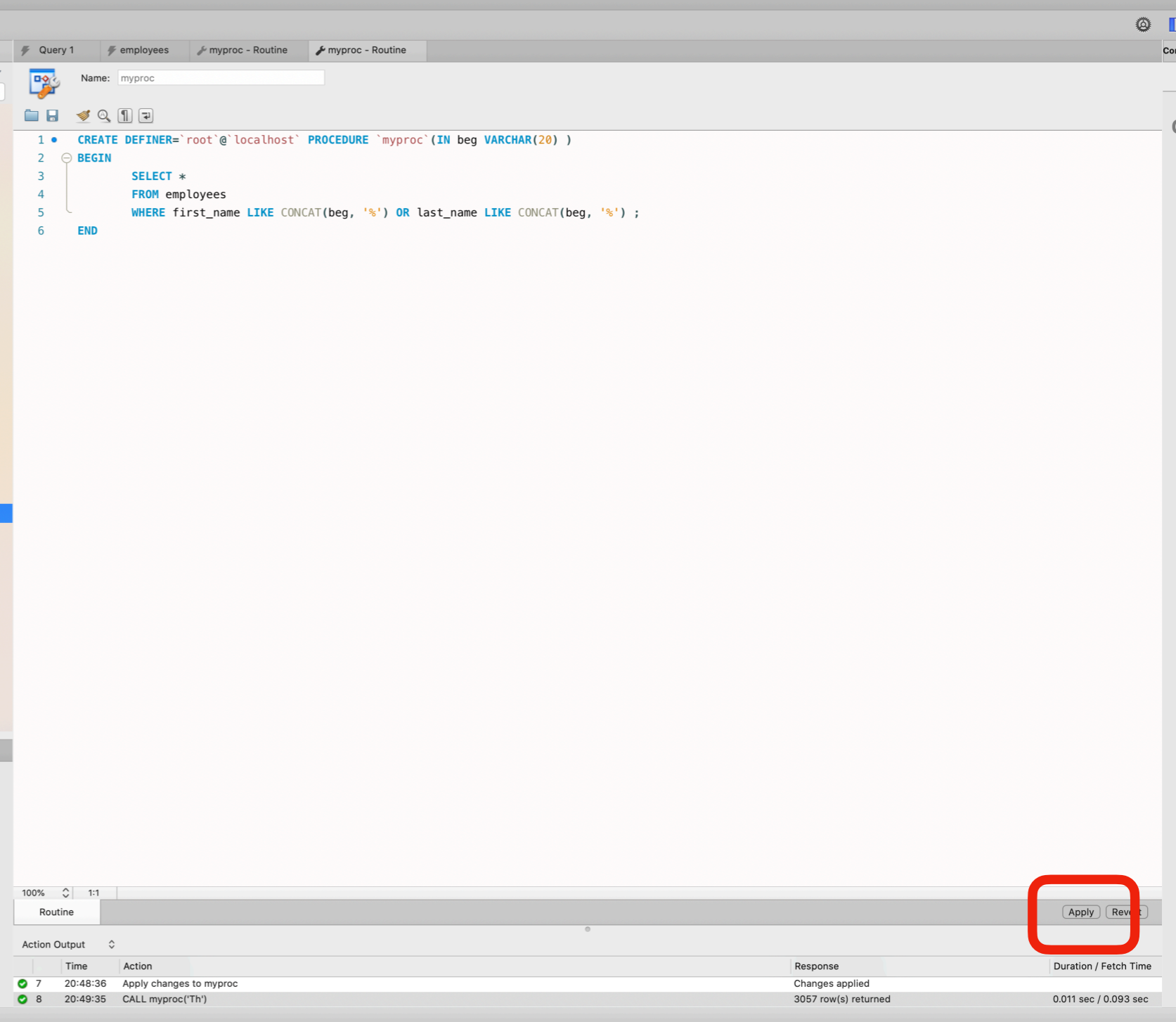
MySQL Stored Procedures

- Example:
 - Change a procedure
 - You can use workbench by selecting the name of the procedure and select 'alter procedure'



MySQL Stored Procedures

- Changing a procedure
- After editing, click Apply



The screenshot shows a MySQL IDE interface with a query editor and an action log. The query editor contains the following SQL code:

```
1 CREATE DEFINER=`root`@`localhost` PROCEDURE `myproc`(IN beg VARCHAR(20) )
2 BEGIN
3     SELECT *
4     FROM employees
5     WHERE first_name LIKE CONCAT(beg, '%') OR last_name LIKE CONCAT(beg, '%') ;
6 END
```

The action log at the bottom shows the following entries:

	Time	Action	Response	Duration / Fetch Time
7	20:48:36	Apply changes to myproc	Changes applied	
8	20:49:35	CALL myproc('Th')	3057 row(s) returned	0.011 sec / 0.093 sec

The 'Apply' button in the bottom right corner of the IDE is highlighted with a red box.

MySQL Stored Procedures

- Changing a procedure:
 - Combine with DROP and CREATE

MySQL Stored Procedures



Single Parameter

```
DELIMITER $$
CREATE PROCEDURE partial_name( IN beg VARCHAR(20) )
BEGIN
    SELECT first_name, last_name, gender
    FROM employees
    WHERE first_name LIKE CONCAT(beg, '%') OR last_name LIKE
CONCAT(beg, '%');
END

DELIMITER ;

CALL partial_name('dan');
```

MySQL Stored Procedures

```
DELIMITER $$
CREATE PROCEDURE partial_name( IN beg VARCHAR(20) )
BEGIN
    SELECT first_name, last_name, gender
    FROM employees
    WHERE first_name LIKE CONCAT(beg, '%') OR last_name LIKE
CONCAT(beg, '%');
END

DELIMITER ;

CALL partial_name('dan');
```

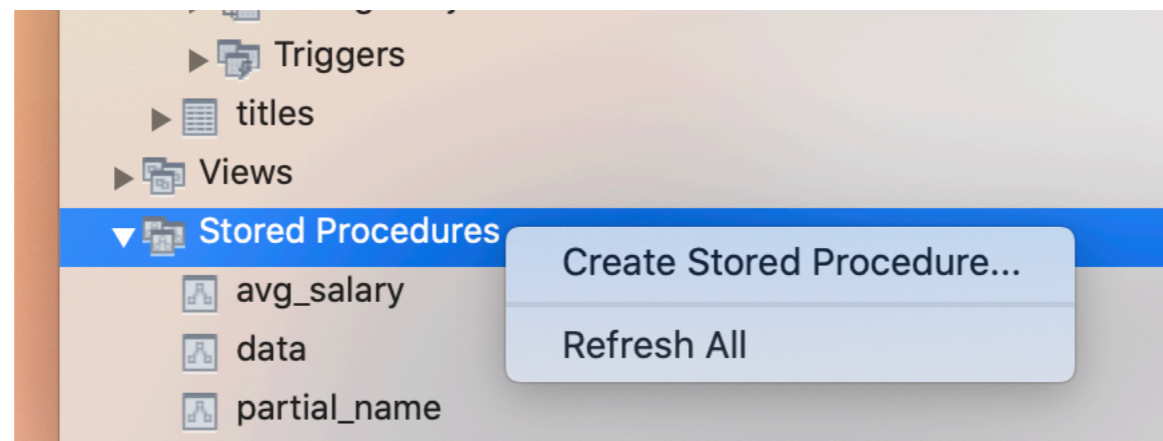
**Have not yet discussed
variables**

MySQL Stored Procedures

- TASK
 - Write a stored procedure that takes as input the first name and the last name of a current employee.
 - It then returns:
 - The first and last name, gender, employee number, department, and last salary of the person (if it is in the database)

HINT

- If you are working with MySQL Workbench, it is less frustrating to define and make changes in the Stored Procedures tab on the left.
- The delimiter statements do not work too well



MySQL Stored Procedures

```
DELIMITER $$
```

```
CREATE PROCEDURE data(IN first VARCHAR(14), IN last VARCHAR(16))  
BEGIN
```

```
    SELECT e.first_name, e.last_name, e.gender, d.dept_name,  
           s.salary
```

```
    FROM employees e, departments d, salaries s, dept_emp de
```

```
    WHERE e.emp_no = de.emp_no AND de.dept_no = d.dept_no
```

```
        AND s.emp_no = e.emp_no AND s.to_date = '9999-01-01'
```

```
        AND e.first_name = first AND e.last_name = last;
```

```
END
```

```
DELIMITER ;
```

MySQL Stored Procedures

- TASK
 - Write a stored procedure that takes as input the first name and the last name of a current or past employee.
 - It then returns:
 - The first and last name, and average salary of the person

MySQL Stored Procedures

```
CREATE PROCEDURE `avg_salary` (  
    IN first VARCHAR(12),  
    last VARCHAR(16)  
)  
BEGIN  
    SELECT e.first_name, e.last_name, AVG(s.salary)  
    FROM employees e, salaries s  
    WHERE e.first_name = first  
        AND e.last_name = last  
        AND e.emp_no = s.emp_no;  
END
```

MySQL Stored Procedures

- Using output variables
 - Let's change the previous procedure to return the average salary
 - We need to use the `SELECT ... INTO ...` construct

MySQL Stored Procedures

```
CREATE PROCEDURE avg_salary(  
    IN first VARCHAR(12),  
    last VARCHAR(16),  
    OUT average_salary DECIMAL(10,2)  
)  
BEGIN  
    SELECT AVG(s.salary)  
    INTO average_salary  
    FROM employees e, salaries s  
    WHERE e.first_name = first  
        AND e.last_name = last  
        AND e.emp_no = s.emp_no;  
END
```



This is our output

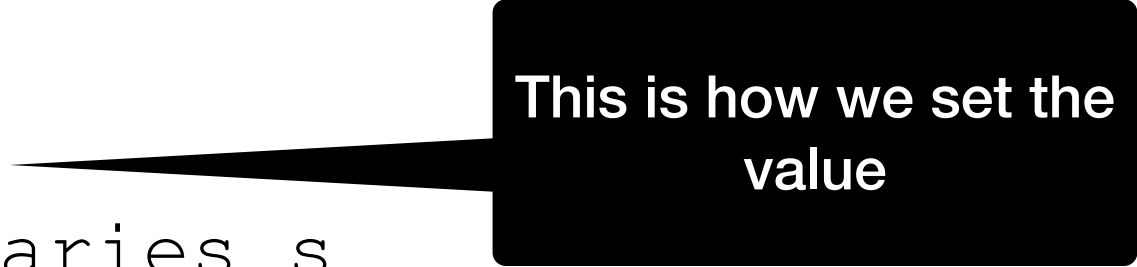
MySQL Stored Procedures

```
CREATE PROCEDURE avg_salary(  
    IN first VARCHAR(12),  
    IN last VARCHAR(16),  
    OUT average_salary DECIMAL(10,2)  
)  
BEGIN  
    SELECT AVG(s.salary)  
    INTO average_salary  
    FROM employees e, salaries s  
    WHERE e.first_name = first  
        AND e.last_name = last  
        AND e.emp_no = s.emp_no;  
END
```

Type is Decimal with
two digits after
comma

MySQL Stored Procedures

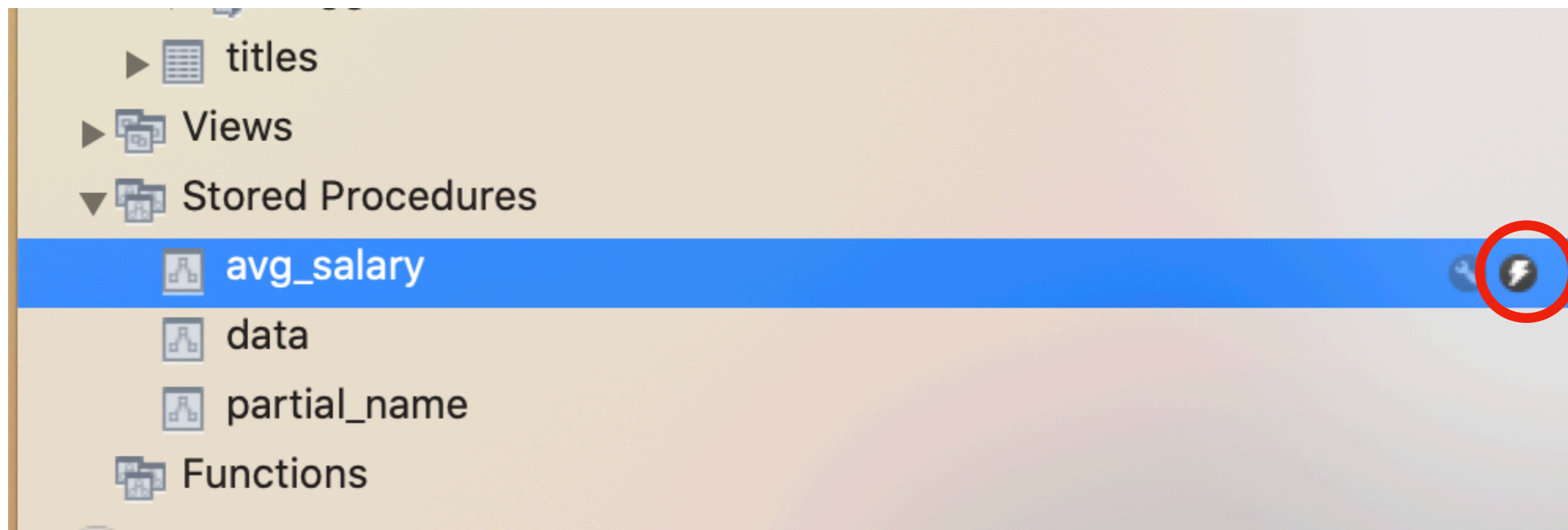
```
CREATE PROCEDURE avg_salary(  
    IN first VARCHAR(12),  
    last VARCHAR(16),  
    OUT average_salary DECIMAL(10,2)  
)  
BEGIN  
    SELECT AVG(s.salary)  
    INTO average_salary  
    FROM employees e, salaries s  
    WHERE e.first_name = first  
        AND e.last_name = last  
        AND e.emp_no = s.emp_no;  
END
```



This is how we set the value

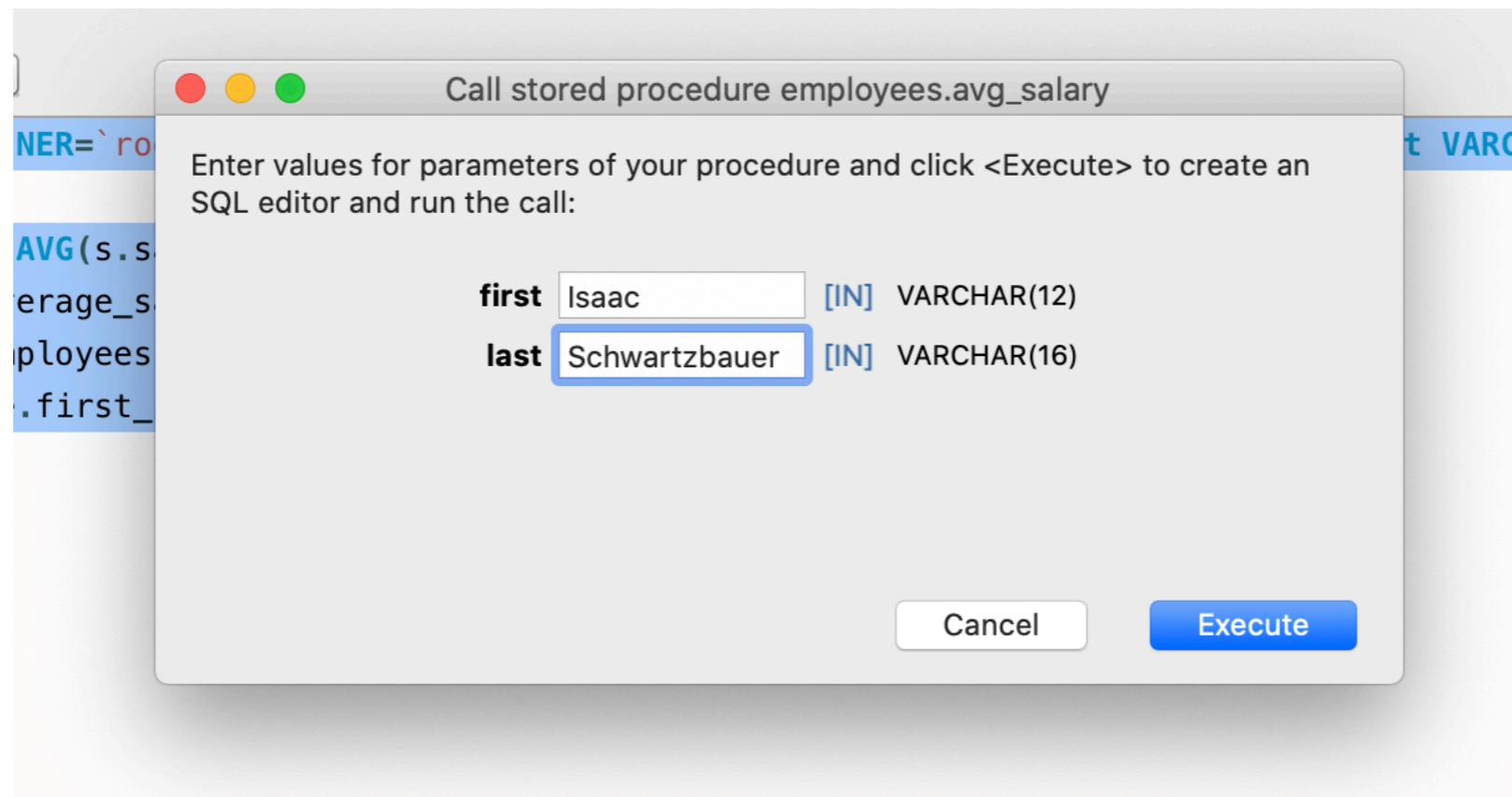
MySQL Stored Procedures

- How do we call this?
 - Easiest is using the lightning symbol in the MySQL work-bench



MySQL Stored Procedures

- This gives you an interactive window



MySQL Stored Procedures

- Result grid shows the value
 - \$75262.06
- But you can also see how to call the procedure as well.

The screenshot shows a MySQL IDE interface. The top pane displays a SQL query with four lines:

```
1 • set @average_salary = 0;  
2 • call employees.avg_salary('Isaac', 'Schwartzbauer', @average_salary);  
3 • select @average_salary;  
4
```

The bottom pane shows the 'Result Grid' with a single column header '@average_salary' and one row containing the value '75262.06'. The interface includes various toolbars and tabs at the top, such as 'Query 1' and 'avg_salary - Routine'.

MySQL Stored Procedures




- Defining variables
 - Variables start with an ampersand @myvar
 - You can enclose the name with ' ' or " " to use other characters than alpha-numeric and '\$'
- Initialized with SET
 - Assignment is with integer, decimal, floating-point, binary or non-binary string, or NULL
 - Can use CAST if necessary

MySQL Stored Procedures

- First, we need to define a variable

```
34 • SET @avgsal = 0;  
35 • CALL avg_salary('Isaac', 'Schwartzbauer', @avgsal);  
36 • SELECT @avgsal;  
37  
38
```

16:36

Result Grid   Filter Rows: Export: 

@avgsal
75262.06

MySQL Stored Procedures

- Solution

```
CREATE DEFINER=`root`@`localhost` PROCEDURE
`avg_salary` (
    IN first VARCHAR(12),
    IN last VARCHAR(16),
    OUT average_salary DECIMAL(10,2)
)
BEGIN
    SELECT AVG(s.salary)
    INTO average_salary
    FROM employees e, salaries s
    WHERE e.first_name = first
        AND e.last_name = last
        AND e.emp_no = s.emp_no;
END
```

MySQL Stored Procedures

- We call the function with

```
SET @avg_sal = 0;  
CALL avg_salary('Isaac', 'Schwartzbauer', @avg_sal);  
SELECT @avg_sal;
```

- Notice how
 - we include the output variable in the call
 - we use SELECT to access the value of the variable

MySQL Stored Procedures

- TASK
 - Write a stored procedure that takes as input the first name and the last name of a current or past employee.
 - It then returns:
 - The employee number
 - Call it from the query tab

MySQL Stored Procedures

- Solution

```
set @empl_numb = 0;
call employees.employee_number(
    'Isaac',
    'Schwartzbauer',
    @empl_numb
);
select @empl_numb;
```

$$x = y \cdot z^2$$

More on Variables

More on Variables

- Definition:

- ```
DECLARE variable_name datatype(size)
 [DEFAULT default_value];
```

- Example:

- ```
DECLARE totalSale DEC(10,2) DEFAULT 0.0;
```

- Recall DEC(10,2) is a decimal number with two decimals after the point.

More on Variables

- Assignment

- Not using the equal-sign =

- Instead SET

- Example:

```
DECLARE total_payments DEC(10,2)  
DEFAULT 0.00;
```

```
SET total_payments = 1521.75;
```

- Or:

```
SELECT SUM(amount)  
INTO total_payments  
FROM payments  
WHERE customerID = 1403;
```

More on Variables

- Scope: “life time of a variable”
 - If a variable is declared within a BEGIN - END block:
 - scope is only the block
 - If a variable name starts with an ampersand “@”:
 - Session variable
 - Goes out of scope only at the end of the session

More on Variables

- Example

```
CREATE PROCEDURE total_order(IN customer_name VARCHAR(16) )
BEGIN
    DECLARE tos INT DEFAULT 0;

    SELECT COUNT(*)
    INTO tos
    FROM orders JOIN customers USING(customerNumber)
    WHERE customerName LIKE CONCAT('%', customer_name, '%');

    SELECT tos;
END
```

MySQL Functions

- A procedure can have more than one output
- Functions have none or one output
- MySQL functions return exactly one value

MySQL Functions

- We can use the MySQL workbench by clicking on Functions in the scheme pane
 - Just below stored procedures
 - Or can use a query
 - Function values are obtained through a SELECT statement
 - `select employees.f_avg(101010);`

MySQL Functions

- Example:

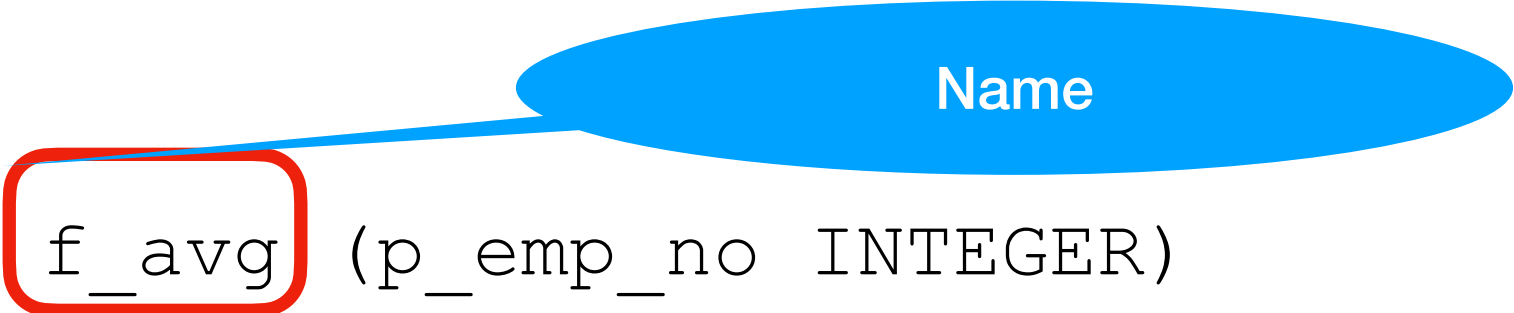
Definition of
a function

```
CREATE FUNCTION f_avg (p_emp_no INTEGER)
RETURNS decimal(10,2)
READS SQL DATA
BEGIN
    DECLARE v_avg_salary DECIMAL(10,2);
    SELECT AVG(salary)
        INTO v_avg_salary
        FROM salaries s
        WHERE s.emp_no = p_emp_no;
    RETURN v_avg_salary;
END
```

MySQL Functions

- Example:

```
CREATE FUNCTION f_avg (p_emp_no INTEGER)
RETURNS decimal(10,2)
READS SQL DATA
BEGIN
    DECLARE v_avg_salary DECIMAL(10,2);
    SELECT AVG(salary)
        INTO v_avg_salary
        FROM salaries s
        WHERE s.emp_no = p_emp_no;
    RETURN v_avg_salary;
END
```



MySQL Functions

- Example:

```
CREATE FUNCTION f_avg (p_emp_no INTEGER)
RETURNS decimal(10,2)
READS SQL DATA
BEGIN
    DECLARE v_avg_salary DECIMAL(10,2);
    SELECT AVG(salary)
        INTO v_avg_salary
        FROM salaries s
        WHERE s.emp_no = p_emp_no;
    RETURN v_avg_salary;
END
```

Name and type of input variable

MySQL Functions

- Example:

```
CREATE FUNCTION f_avg (p_emp_no INTEGER)
RETURNS decimal(10,2)
READS SQL DATA
BEGIN
    DECLARE v_avg_salary DECIMAL(10,2);
    SELECT AVG(salary)
        INTO v_avg_salary
        FROM salaries s
        WHERE s.emp_no = p_emp_no;
    RETURN v_avg_salary;
END
```

Keyword Returns is needed
plus specification of return
value type

MySQL Functions

- Example:

```
CREATE FUNCTION f_avg (p_emp_no INTEGER)
RETURNS decimal(10,2)
READS SQL DATA
BEGIN
    DECLARE v_avg_salary DECIMAL(10,2);
    SELECT AVG(salary)
        INTO v_avg_salary
        FROM salaries s
        WHERE s.emp_no = p_emp_no;
    RETURN v_avg_salary;
END
```

Describes behavior of the function

MySQL Functions

- DETERMINISTIC:
 - always produces the same result for the same parameter
- NO SQL:
 - function has no SQL statements
- READS SQL DATA
 - contains statements that read via SQL, but does not modify the database
- MODIFIES SQL DATA
 - contains statements that write (e.g. inserts)

MySQL Functions

- Example:

Body of the function

```
CREATE FUNCTION f_avg (p_emp_no INTEGER)
RETURNS decimal(10,2)
READS SQL DATA
BEGIN
    DECLARE v_avg_salary DECIMAL(10,2);
    SELECT AVG(salary)
        INTO v_avg_salary
        FROM salaries s
        WHERE s.emp_no = p_emp_no;
    RETURN v_avg_salary;
END
```

MySQL Functions

- Example:

Here we define a variable to be used in several statements

```
CREATE FUNCTION f_avg (p_emp_no INTEGER)
RETURNS decimal(10,2)
READS SQL DATA
BEGIN
```

```
    DECLARE v_avg_salary DECIMAL(10,2);
```

```
    SELECT AVG(salary)
        INTO v_avg_salary
        FROM salaries s
        WHERE s.emp_no = p_emp_no;
    RETURN v_avg_salary;
```

```
END
```

MySQL Functions

- Example:

The variable has type Decimal(10,2) to represent currency values

```
CREATE FUNCTION f_avg (p_emp_no INTEGER)
RETURNS decimal(10,2)
READS SQL DATA
BEGIN
    DECLARE v_avg_salary DECIMAL(10,2);
    SELECT AVG(salary)
        INTO v_avg_salary
        FROM salaries s
        WHERE s.emp_no = p_emp_no;
    RETURN v_avg_salary;
END
```

MySQL Functions

- Example:

The SELECT ... INTO ... clause updates the value of v_avg_salary

```
CREATE FUNCTION f_avg (p_emp_no INTEGER)
RETURNS decimal(10,2)
READS SQL DATA
BEGIN
    DECLARE v_avg_salary DECIMAL(10,2);
    SELECT AVG(salary)
        INTO v_avg_salary
        FROM salaries s
        WHERE s.emp_no = p_emp_no;
    RETURN v_avg_salary;
END
```

MySQL Functions

- Example:

A function has to return a value

```
CREATE FUNCTION f_avg (p_emp_no INTEGER)
RETURNS decimal(10,2)
READS SQL DATA
BEGIN
    DECLARE v_avg_salary DECIMAL(10,2);
    SELECT AVG(salary)
        INTO v_avg_salary
        FROM salaries s
        WHERE s.emp_no = p_emp_no;
    RETURN v_avg_salary;
END
```


MySQL Functions

- Functions and procedures can have more than one select value
 - Find the last salary of an employee given by first and last name
 - Query 1: Find the last from_date in salaries for the employee
 - We store it in a variable
 - Query 2: Return the corresponding salary

```
CREATE FUNCTION f_latest_salary(p_first_name VARCHAR(12), p_last_name VARCHAR(16))
    RETURNS decimal(10,2)
    READS SQL DATA
BEGIN
    DECLARE v_max_from_date date;
    DECLARE v_last_salary DECIMAL(10,2);

    SELECT
        MAX(from_date)
    INTO v_max_from_date
    FROM employees e
    JOIN salaries s ON e.emp_no = s.emp_no
    WHERE e.first_name = p_first_name
        AND e.last_name = p_last_name;

    SELECT
        s.salary
    INTO v_last_salary
    FROM employees e
    JOIN salaries s ON e.emp_no = s.emp_no
    WHERE e.first_name = p_first_name
        AND e.last_name = p_last_name
        AND s.from_date = v_max_from_date;

    RETURN v_last_salary;
END
```

```
CREATE FUNCTION f_latest_salary(p_first_name VARCHAR(12), p_last_name VARCHAR(16))
    RETURNS decimal(10,2)
    READS SQL DATA
BEGIN
    DECLARE v_max_from_date date;
    DECLARE v_last_salary DECIMAL(10,2);

    SELECT
        MAX(from_date)
    INTO v_max_from_date
    FROM employees e
    JOIN salaries s ON e.emp_no = s.emp_no
    WHERE e.first_name = p_first_name
        AND e.last_name = p_last_name;

    SELECT
        s.salary
    INTO v_last_salary
    FROM employees e
    JOIN salaries s ON e.emp_no = s.emp_no
    WHERE e.first_name = p_first_name
        AND e.last_name = p_last_name
        AND s.from_date = v_max_from_date;

    RETURN v_last_salary;
END
```

**Declare a variable of
type date to contain the
last contract to-date**

```
CREATE FUNCTION f_latest_salary(p_first_name VARCHAR(12), p_last_name VARCHAR(16))
    RETURNS decimal(10,2)
    READS SQL DATA
BEGIN
    DECLARE v_max_from_date date;
    DECLARE v_last_salary DECIMAL(10,2);

    SELECT MAX(from_date)
    INTO v_max_from_date
    FROM employees e
    JOIN salaries s ON e.emp_no = s.emp_no
    WHERE e.first_name = p_first_name
        AND e.last_name = p_last_name;

    SELECT
        s.salary
    INTO v_last_salary
    FROM employees e
    JOIN salaries s ON e.emp_no = s.emp_no
    WHERE e.first_name = p_first_name
        AND e.last_name = p_last_name
        AND s.from_date = v_max_from_date;

    RETURN v_last_salary;
END
```

**Declare a variable of
type Decimal for the
salary**

```


CREATE FUNCTION f_latest_salary(p_first_name VARCHAR(12), p_last_name VARCHAR(16))
  RETURNS decimal(10,2)
  READS SQL DATA
BEGIN
  DECLARE v_max_from_date date;
  DECLARE v_last_salary DECIMAL(10,2);

  SELECT MAX(from_date)
  INTO v_max_from_date
  FROM employees e
  JOIN salaries s ON e.emp_no = s.emp_no
  WHERE e.first_name = p_first_name
         AND e.last_name = p_last_name;

  SELECT
    s.salary
  INTO v_last_salary
  FROM employees e
  JOIN salaries s ON e.emp_no = s.emp_no
  WHERE e.first_name = p_first_name
         AND e.last_name = p_last_name
         AND s.from_date = v_max_from_date;

  RETURN v_last_salary;
END

```



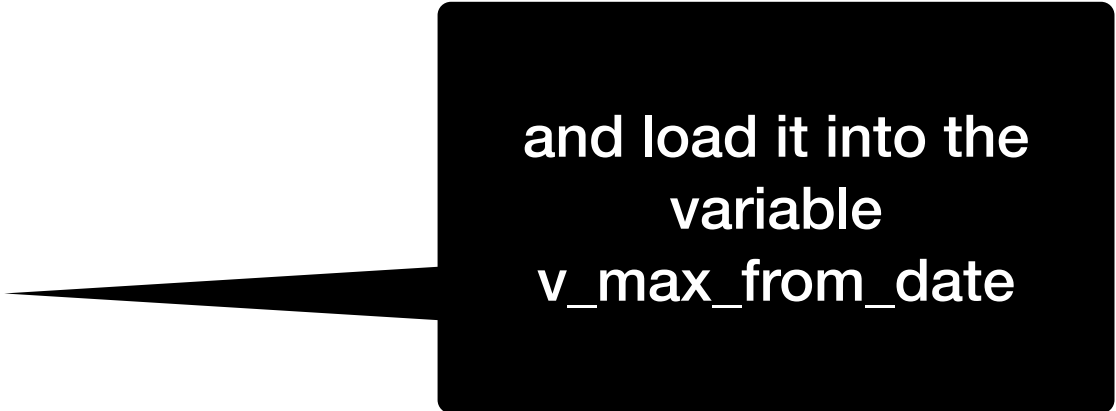
Determine the last contract from_date for an employee with given first and last name

```
CREATE FUNCTION f_latest_salary(p_first_name VARCHAR(12), p_last_name VARCHAR(16))
  RETURNS decimal(10,2)
  READS SQL DATA
BEGIN
  DECLARE v_max_from_date date;
  DECLARE v_last_salary DECIMAL(10,2);

  SELECT MAX(from_date)
  INTO v_max_from_date
  FROM employees e
  JOIN salaries s ON e.emp_no = s.emp_no
  WHERE e.first_name = p_first_name
         AND e.last_name = p_last_name;

  SELECT
    s.salary
  INTO v_last_salary
  FROM employees e
  JOIN salaries s ON e.emp_no = s.emp_no
  WHERE e.first_name = p_first_name
         AND e.last_name = p_last_name
         AND s.from_date = v_max_from_date;

  RETURN v_last_salary;
END
```



and load it into the
variable
v_max_from_date

```
CREATE FUNCTION f_latest_salary(p_first_name VARCHAR(12), p_last_name VARCHAR(16))
    RETURNS decimal(10,2)
    READS SQL DATA
BEGIN
    DECLARE v_max_from_date date;
    DECLARE v_last_salary DECIMAL(10,2);

    SELECT MAX(from_date)
    INTO v_max_from_date
    FROM employees e
    JOIN salaries s ON e.emp_no = s.emp_no
    WHERE e.first_name = p_first_name
        AND e.last_name = p_last_name;

    SELECT
        s.salary
    INTO v_last_salary
    FROM employees e
    JOIN salaries s ON e.emp_no = s.emp_no
    WHERE e.first_name = p_first_name
        AND e.last_name = p_last_name
        AND s.from_date = v_max_from_date;

    RETURN v_last_salary;
END
```

Second query:


**Find the corresponding
amount**

```
CREATE FUNCTION f_latest_salary(p_first_name VARCHAR(12), p_last_name VARCHAR(16))
    RETURNS decimal(10,2)
    READS SQL DATA
BEGIN
    DECLARE v_max_from_date date;
    DECLARE v_last_salary DECIMAL(10,2);

    SELECT MAX(from_date)
    INTO v_max_from_date
    FROM employees e
    JOIN salaries s ON e.emp_no = s.emp_no
    WHERE e.first_name = p_first_name
        AND e.last_name = p_last_name;

    SELECT
        s.salary
    INTO v_last_salary
    FROM employees e
    JOIN salaries s ON e.emp_no = s.emp_no
    WHERE e.first_name = p_first_name
        AND e.last_name = p_last_name
        AND s.from_date = v_max_from_date;

    RETURN v_last_salary;
END
```



**And finally return this
value**

MySQL Functions

- Why do we need functions?
 - Procedures can not be embedded in a select statement
 - But functions can

MySQL Variables

- MySQL has three types of variables
 - Local: Scope is in a BEGIN ... END block
 - Use DECLARE and no ampersand
- Session
 - Starts and ends with making a connection to the database
 - One session per current user
 - Use SET @variable = NULL
- Global

MySQL Variables

- Global variables
 - Survives disconnection
 - Needs to be system variables
 - E.g. `.max_connections()`, `.max_join_size()`
 - `SET GLOBAL max_connections = 1000;`
 - `SET @@global.max_connections = 1000;`

MySQL Variables

- Try it out
 - Set `max_connections` to 1
 - Then try another connection in MySQL workbench

Conditions

- MySQL has an IF statement
- MySQL has a CASE statement
- My SQL has a CASE expression
 - which is the easiest

```
CASE attribute
  WHEN ... THEN ...
  ELSE ...
END
```

CASE Statement

- Example
 - Expanding gender

```
SELECT first_name, last_name, CASE
      WHEN 'M' THEN 'male'
      ELSE 'female'
      END AS gender
FROM employees
WHERE emp_id = p_emp_id;
```

CASE Statement

- Example:
 - Select employee data, including whether they are managers or not
 - Information is in employees and in dept_manager table
 - One possibility: Use a LEFT JOIN
 - Also: limit queries to a range of emp_no chosen to have managers and no-managers in it

CASE Statement

```
SELECT e.emp_no,  
       e.first_name,  
       e.last_name,  
       dm.emp_no  
FROM employees e LEFT JOIN dept_manager dm  
                 ON e.emp_no = dm.emp_no  
WHERE e.emp_no BETWEEN 109990 AND 111000;
```


CASE Statement

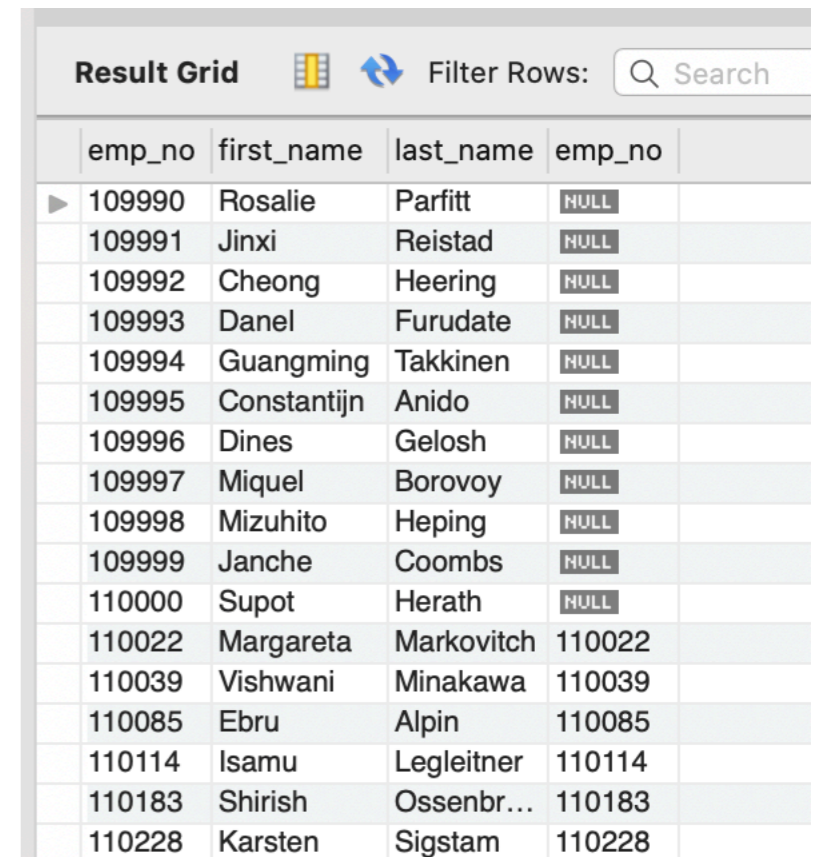
```
SELECT e.emp_no,  
       e.first_name,  
       e.last_name,  
       dm.emp_no  
FROM employees e LEFT  
       ON e.emp_no  
WHERE e.emp_no BETWEEN
```

emp_no	first_name	last_name	emp_no
109990	Rosalie	Parfitt	NULL
109991	Jinxi	Reistad	NULL
109992	Cheong	Heering	NULL
109993	Danel	Furudate	NULL
109994	Guangming	Takkinen	NULL
109995	Constantijn	Anido	NULL
109996	Dines	Gelosh	NULL
109997	Miquel	Borovoy	NULL
109998	Mizuhito	Heping	NULL
109999	Janche	Coombs	NULL
110000	Supot	Herath	NULL
110022	Margareta	Markovitch	110022
110039	Vishwani	Minakawa	110039
110085	Ebru	Alpin	110085
110114	Isamu	Legleitner	110114
110183	Shirish	Ossenbr...	110183
110228	Karsten	Sigstam	110228

per dm
000;

CASE Statement

- Because this is a left join, we match the emp_no, but keep both of them
 - the second one is for dept_manager
- This means that we can use the case statement



The screenshot shows a 'Result Grid' interface with a search bar and a refresh icon. The table displays the results of a left join between an employee table and a manager table. The columns are emp_no, first_name, last_name, and emp_no. The first 10 rows show employees with no matching manager (NULL). The last 6 rows show employees who are also managers, with their manager emp_no filled in.

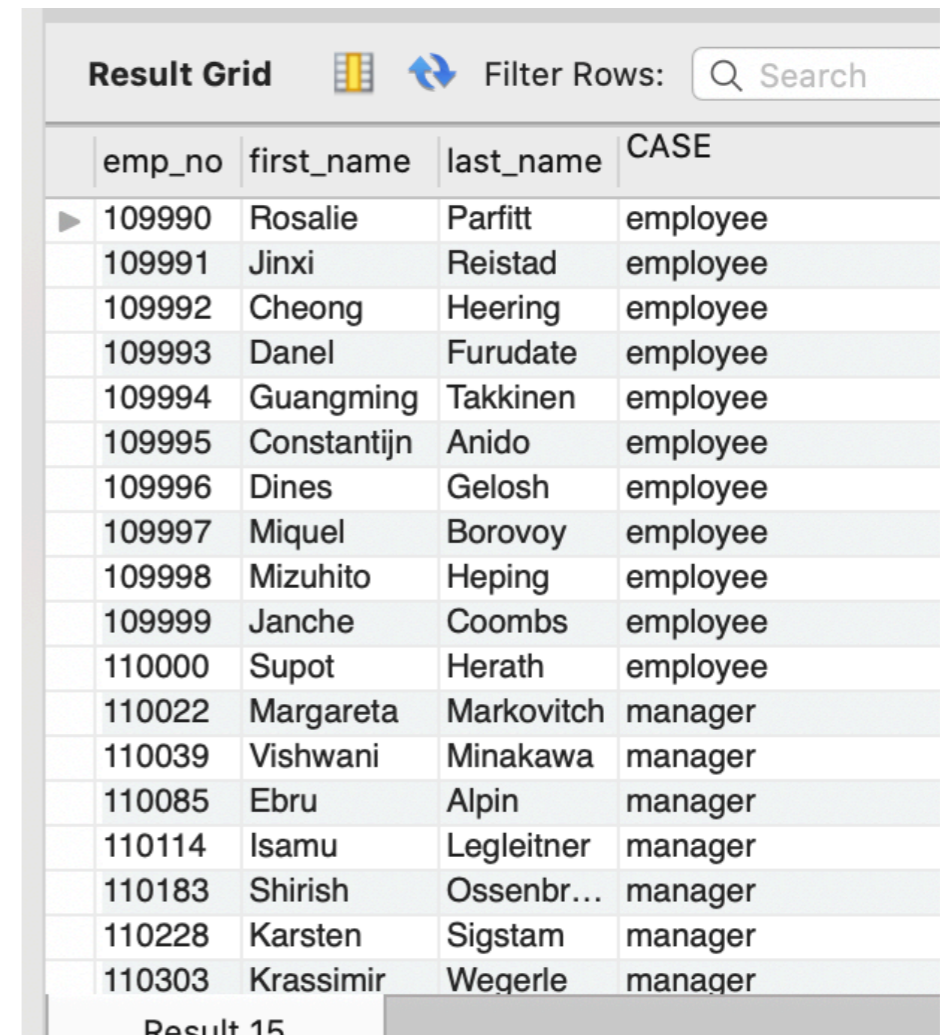
emp_no	first_name	last_name	emp_no
109990	Rosalie	Parfitt	NULL
109991	Jinxi	Reistad	NULL
109992	Cheong	Heering	NULL
109993	Danel	Furudate	NULL
109994	Guangming	Takkinen	NULL
109995	Constantijn	Anido	NULL
109996	Dines	Gelosh	NULL
109997	Miquel	Borovoy	NULL
109998	Mizuhito	Heping	NULL
109999	Janche	Coombs	NULL
110000	Supot	Herath	NULL
110022	Margareta	Markovitch	110022
110039	Vishwani	Minakawa	110039
110085	Ebru	Alpin	110085
110114	Isamu	Legleitner	110114
110183	Shirish	Ossenbr...	110183
110228	Karsten	Sigstam	110228



CASE Statement

```
SELECT e.emp_no,  
       e.first_name,  
       e.last_name,  
       CASE  
           WHEN dm.emp_no IS NOT NULL THEN 'manager'  
           ELSE 'employee'  
       END  
FROM employees e LEFT JOIN dept_manager dm  
                 ON e.emp_no = dm.emp_no  
WHERE e.emp_no BETWEEN 109990 AND 111000;
```

CASE Statement

- The last column of the result is now the result of the CASE expression
- The column name is bad, so we change it with an AS clause



Result Grid   Filter Rows:

emp_no	first_name	last_name	CASE
109990	Rosalie	Parfitt	employee
109991	Jinxi	Reistad	employee
109992	Cheong	Heering	employee
109993	Danel	Furudate	employee
109994	Guangming	Takkinen	employee
109995	Constantijn	Anido	employee
109996	Dines	Gelosh	employee
109997	Miquel	Borovoy	employee
109998	Mizuhito	Heping	employee
109999	Janche	Coombs	employee
110000	Supot	Herath	employee
110022	Margareta	Markovitch	manager
110039	Vishwani	Minakawa	manager
110085	Ebru	Alpin	manager
110114	Isamu	Legleitner	manager
110183	Shirish	Ossenbr...	manager
110228	Karsten	Sigstam	manager
110303	Krassimir	Wegerle	manager

Result 15

CASE Statement

```
SELECT e.emp_no,  
       e.first_name,  
       e.last_name,  
       CASE  
           WHEN dm.emp_no IS NOT NULL THEN 'manager'  
           ELSE 'employee'  
       END AS 'role'  
FROM employees e LEFT JOIN dept_manager dm  
       ON e.emp_no = dm.emp_no  
WHERE e.emp_no BETWEEN 109990 AND 111000;
```

CASE Statement

```
SELECT e.emp_no,  
       e.first_name,  
       e.last_name,  
       CASE  
           WHEN dm.emp_no IS NULL  
           THEN 'employee'  
           ELSE 'employee'  
       END AS 'role'  
FROM employees e LEFT JOIN departments dm  
    ON e.emp_no = dm.emp_no  
WHERE e.emp_no BETWEEN 109990
```

Result Grid					Filter Rows:	Search	Export:
	emp_no	first_name	last_name	role			
▶	109990	Rosalie	Parfitt	employee			
	109991	Jinxi	Reistad	employee			
	109992	Cheong	Heering	employee			
	109993	Danel	Furudate	employee			
	109994	Guangming	Takkinen	employee			
	109995	Constantijn	Anido	employee			
	109996	Dines	Gelosh	employee			
	109997	Miquel	Borovoy	employee			
	109998	Mizuhito	Heping	employee			
	109999	Janche	Coombs	employee			
	110000	Supot	Herath	employee			
	110022	Margareta	Markovitch	manager			
	110039	Vishwani	Minakawa	manager			
	110085	Ebru	Alpin	manager			
	110114	Isamu	Legleitner	manager			
	110183	Shirish	Ossenbrug...	manager			
	110228	Karsten	Sigstam	manager			
	110303	Krassimir	Wegerle	manager			
	110344	Rosine	Cools	manager			
	110386	Shem	Kieras	manager			
	110420	Oscar	Ghazalie	manager			
	110511	DeForest	Hagimont	manager			
	110567	Leon	DasSarma	manager			
	110725	Peternela	Onuegbe	manager			
	110765	Rutger	Hofmeyr	manager			
	110800	Sanjoy	Quadeer	manager			
	110854	Dung	Pesch	manager			

CASE Statement

- **GOTCHA**

- NULL and NOT NULL cannot be compared

- So: this does NOT work

```
SELECT e.emp_no,  
       e.first_name,  
       e.last_name,  
       CASE dm.emp_no  
         WHEN NOT NULL THEN 'manager'  
         ELSE 'employee'  
       END AS 'role'  
FROM employees e LEFT JOIN dept_manager dm  
  ON e.emp_no = dm.emp_no  
WHERE e.emp_no BETWEEN 109990 AND 111000;
```

CASE Statement

- Result:
 - Everyone is an employee because NOT NULL comparison is never-ever true

emp_no	first_name	last_name	role
▶ 109990	Rosalie	Parfitt	employee
109991	Jinxi	Reistad	employee
109992	Cheong	Heering	employee
109993	Danel	Furudate	employee
109994	Guangming	Takkinen	employee
109995	Constantijn	Anido	employee
109996	Dines	Gelosh	employee
109997	Miquel	Borovoy	employee
109998	Mizuhito	Heping	employee
109999	Janche	Coombs	employee
110000	Supot	Herath	employee
110022	Margareta	Markovitch	employee
110039	Vishwani	Minakawa	employee
110085	Ebru	Alpin	employee
110114	Isamu	Legleitner	employee
110183	Shirish	Ossenbrug...	employee
110228	Karsten	Sigstam	employee
110303	Krassimir	Wegerle	employee
110344	Rosine	Cools	employee
110386	Shem	Kieras	employee
110420	Oscar	Ghazalie	employee
110511	DeForest	Hagimont	employee
110567	Leon	DasSarma	employee
110725	Peternela	Onuegbe	employee
110765	Rutger	Hofmeyr	employee

CASE Statement

- MySQL has an IF expression
 - Can have only one test
 - Syntax is IF(condition, valiftrue, valiffalse)

CASE Statement

- Inside a stored procedure, we can use the case statement to set a variable

```
CREATE PROCEDURE GetDeliveryStatus(
  IN pOrderNumber INT,
  OUT pDeliveryStatus VARCHAR(100)
)
BEGIN
  DECLARE waitingDay INT DEFAULT 0;
  SELECT
    DATEDIFF(requiredDate, shippedDate)
  INTO waitingDay
  FROM orders
  WHERE orderNumber = pOrderNumber;

  CASE
    WHEN waitingDay = 0 THEN
      SET pDeliveryStatus = 'On Time';
    WHEN waitingDay >= 1 AND waitingDay < 5 THEN
      SET pDeliveryStatus = 'Late';
    WHEN waitingDay >= 5 THEN
      SET pDeliveryStatus = 'Very Late';
    ELSE
      SET pDeliveryStatus = 'No Information';
  END CASE;
```