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

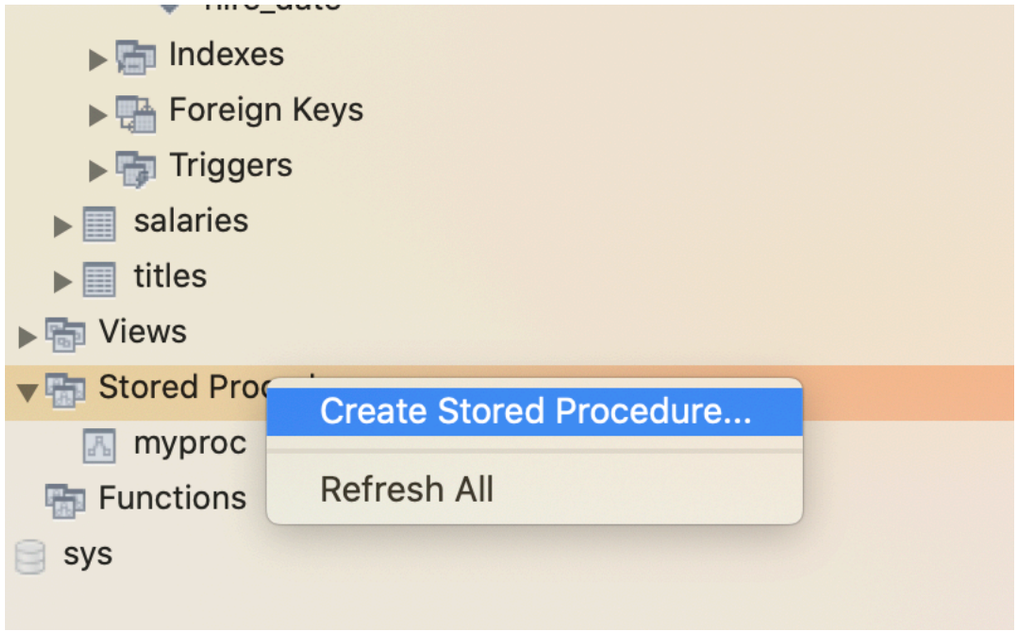
MySQL Stored Procedures

- 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 $$
```

Need to give it a name

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

DELIMITER ;
```

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

- 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
```

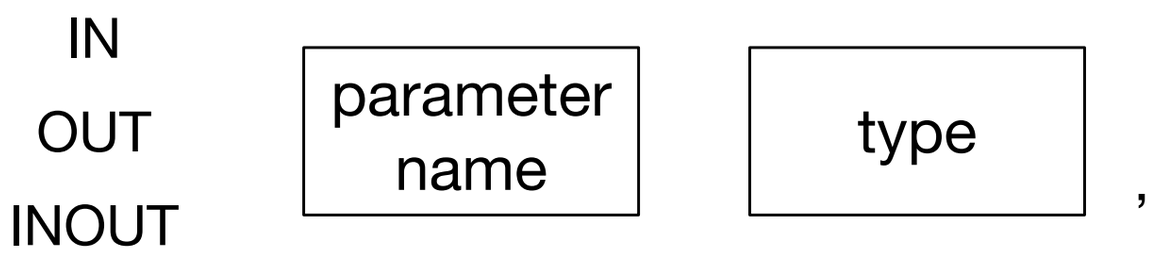
100% 15:14

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

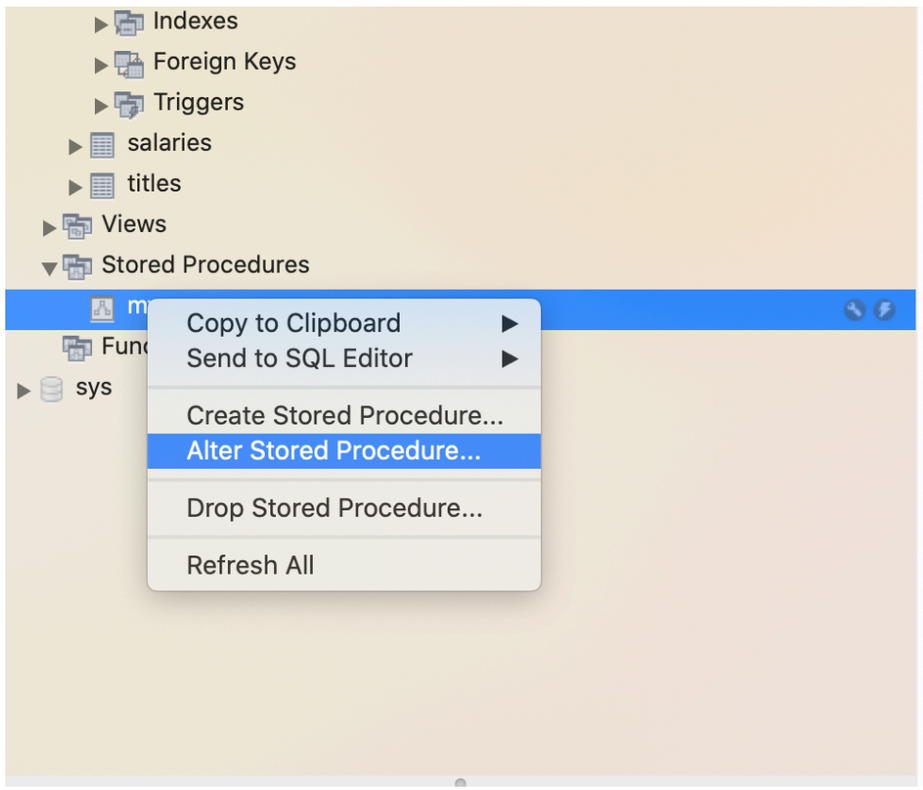
MySQL Stored Procedures

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



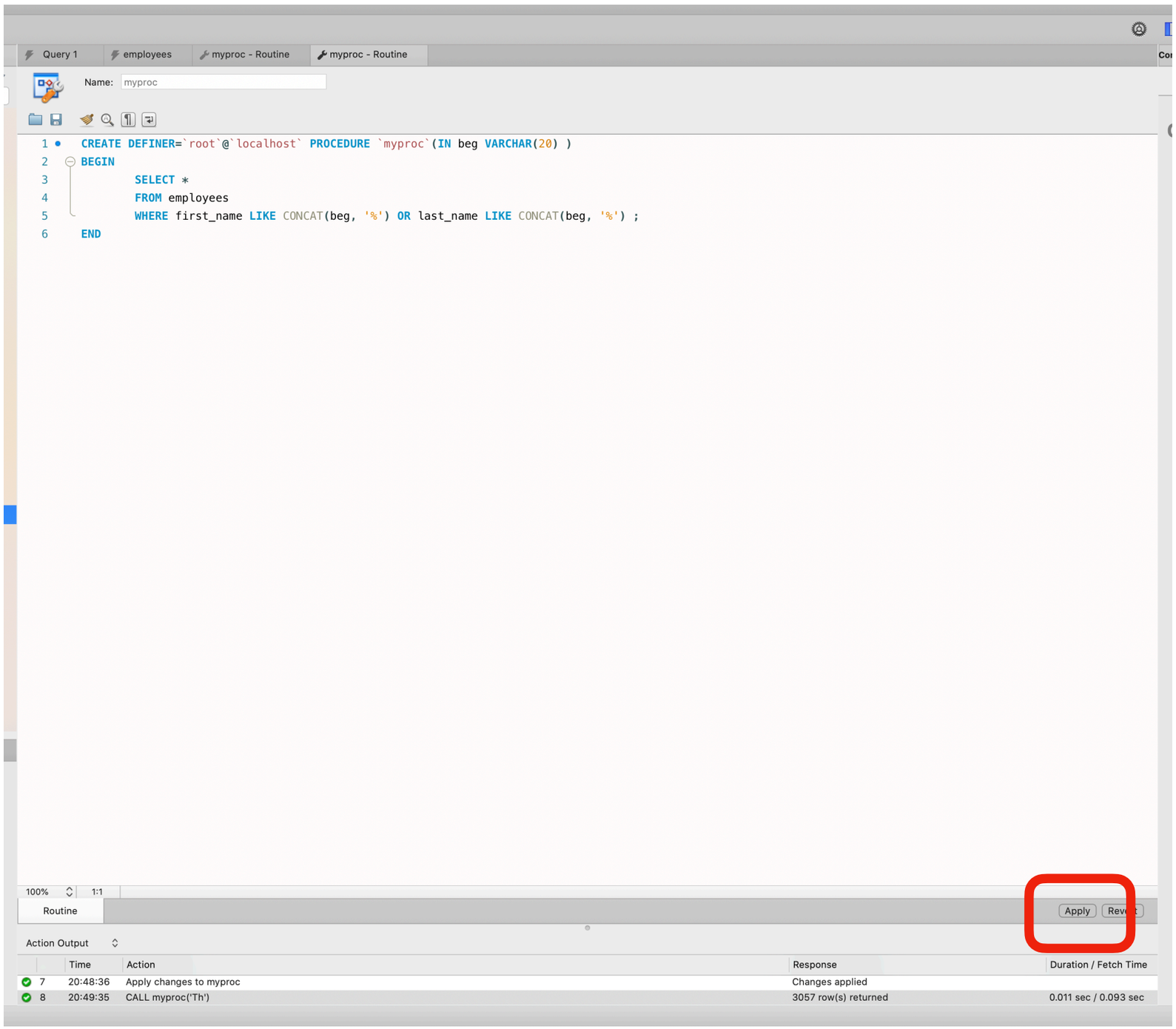
MySQL Stored Procedures

- 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



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');
```

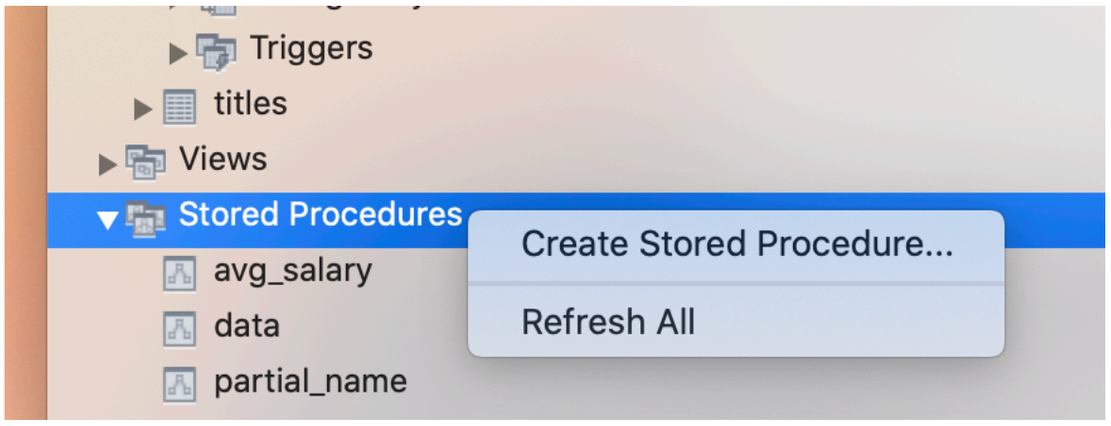
Have not yet discussed variables

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),
        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

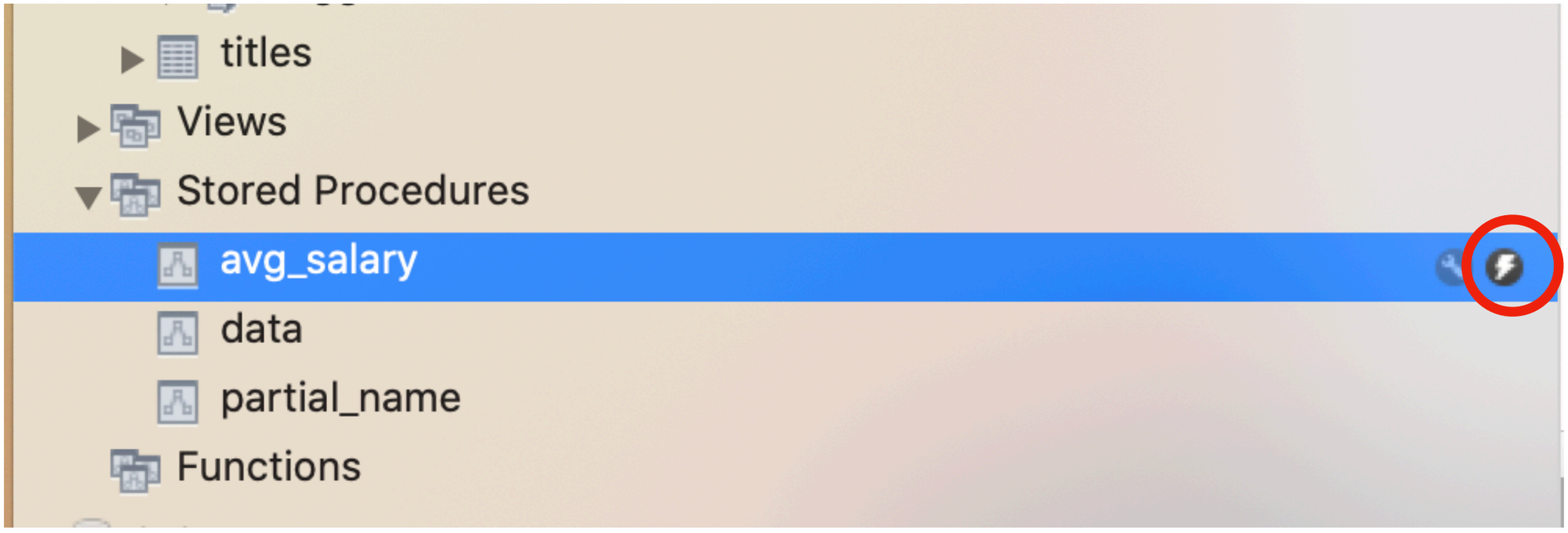
MySQL Stored Procedures

```
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

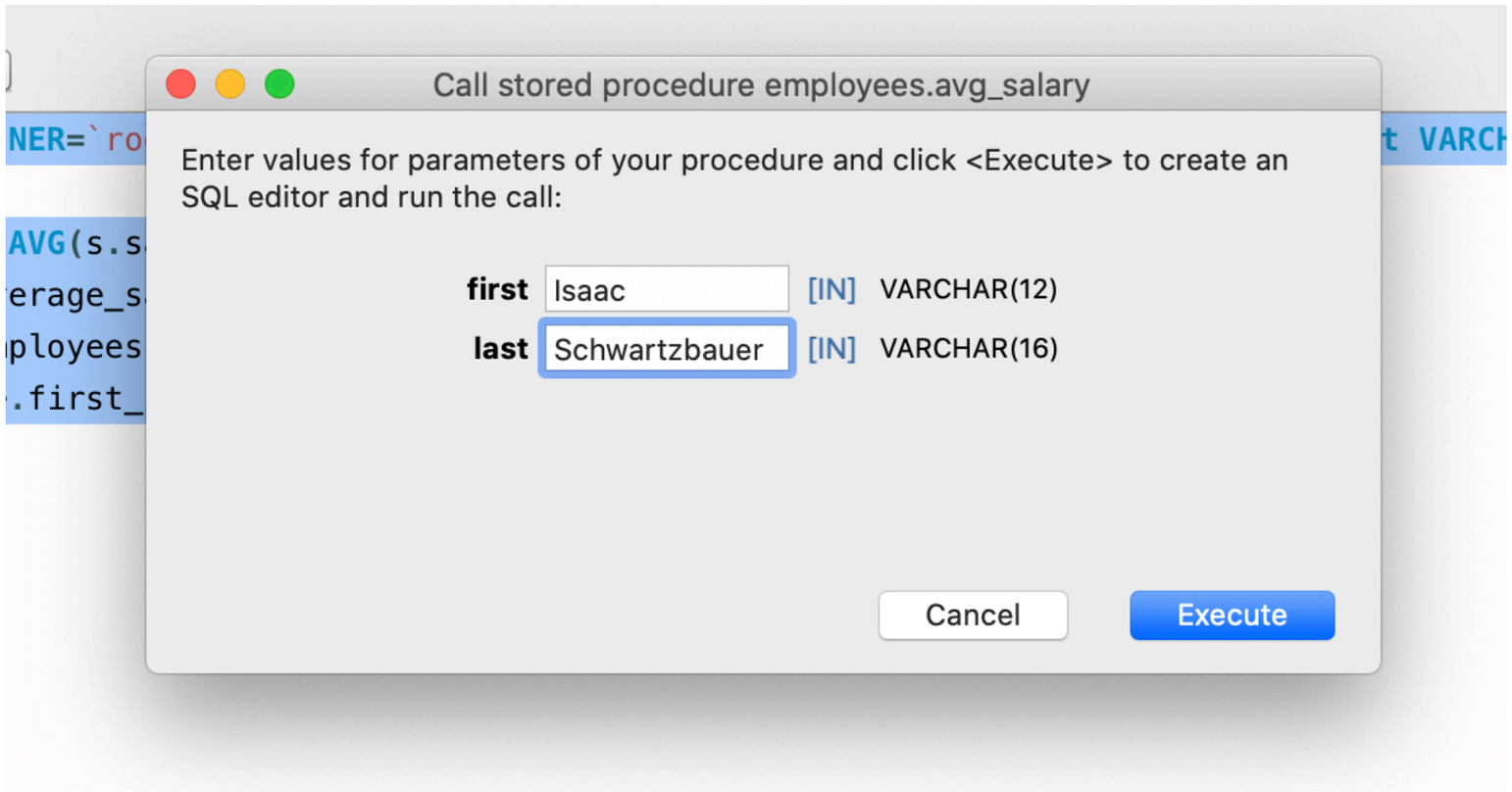
MySQL Stored Procedures

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



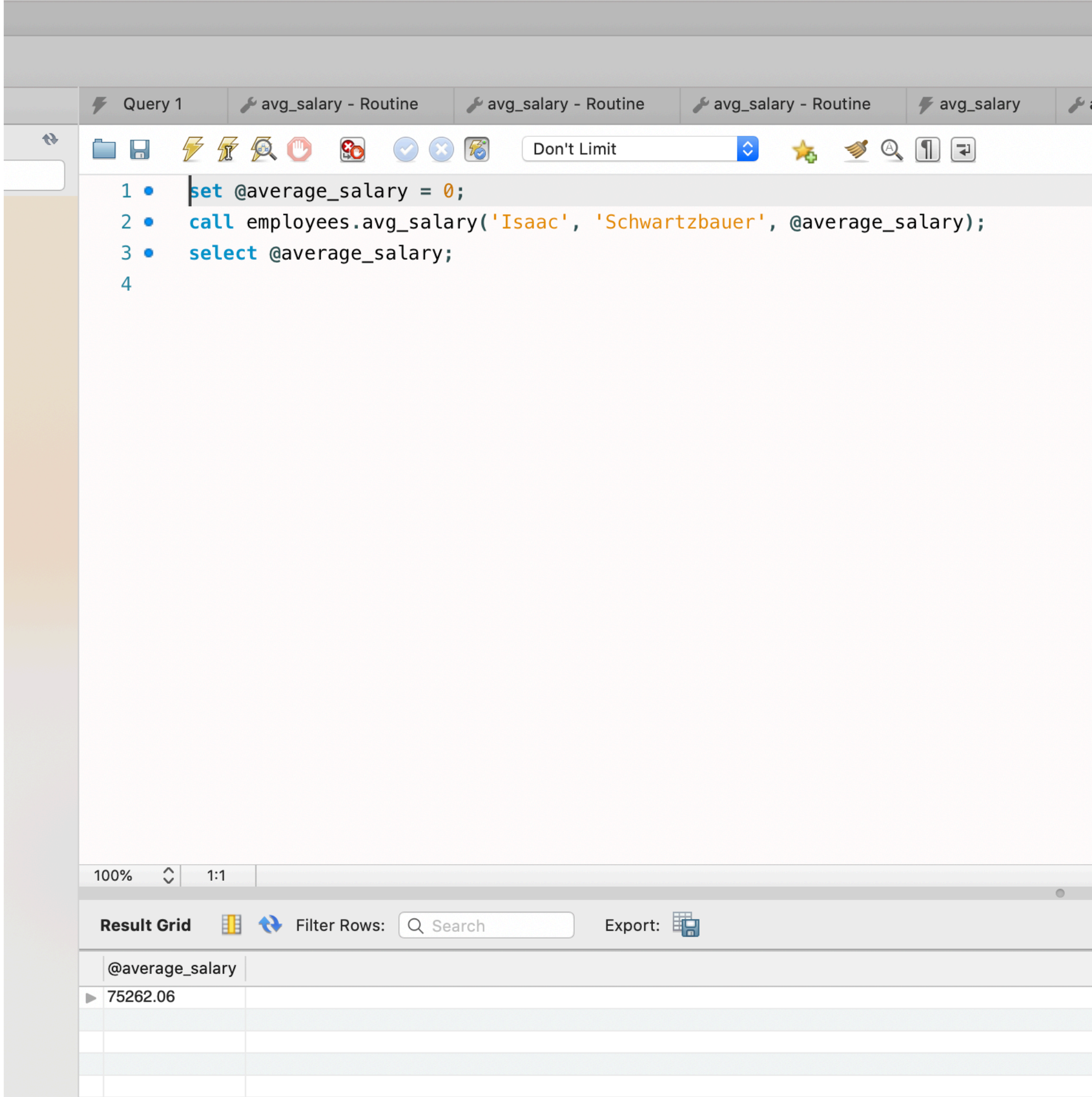
MySQL Stored Procedures

- 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.
-



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
```

16:36

Result Grid Filter Rows: Search Export:

@avgsal
75262.06

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

- We call the function with

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

- 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

```
CREATE DEFINER=`root`@`localhost` PROCEDURE
`employee_number`(
   IN first VARCHAR(12),
   IN last VARCHAR(16),
   OUT empl_numb INT
  )
BEGIN
   SELECT emp_no
     INTO empl_numb
     FROM employees
     WHERE first_name = first
        AND last_name = last;
END
```

MySQL Stored Procedures

- Solution

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

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:

Name and type of input variable

```
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:

Keyword Returns is needed plus specification of return value type

```
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
```

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_salary (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_
WHERE e.emp_no BETWEE
```

Result Grid | Filter Rows: Search

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

- 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

Result Grid Filter Rows: Search

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: Search

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 I
           ELSE 'employee'
       END AS 'role'
FROM employees e LEFT JOIN de
           ON e.emp_no = dm.e
WHERE e.emp_no BETWEEN 10999(
```

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;
```