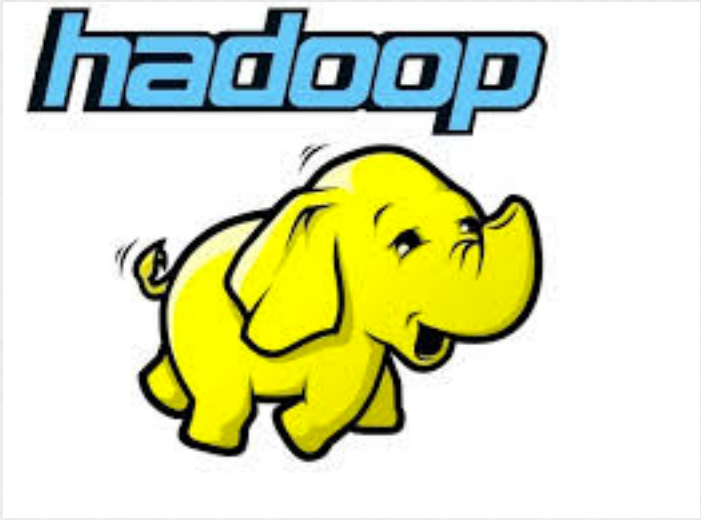
UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS



BIG DATA
Analytics & Management

Lecture 5 (02/28, 03/05): Advanced SQL

Decisions, Operations & Information Technologies
Robert H. Smith School of Business
Spring, 2018

Procedure

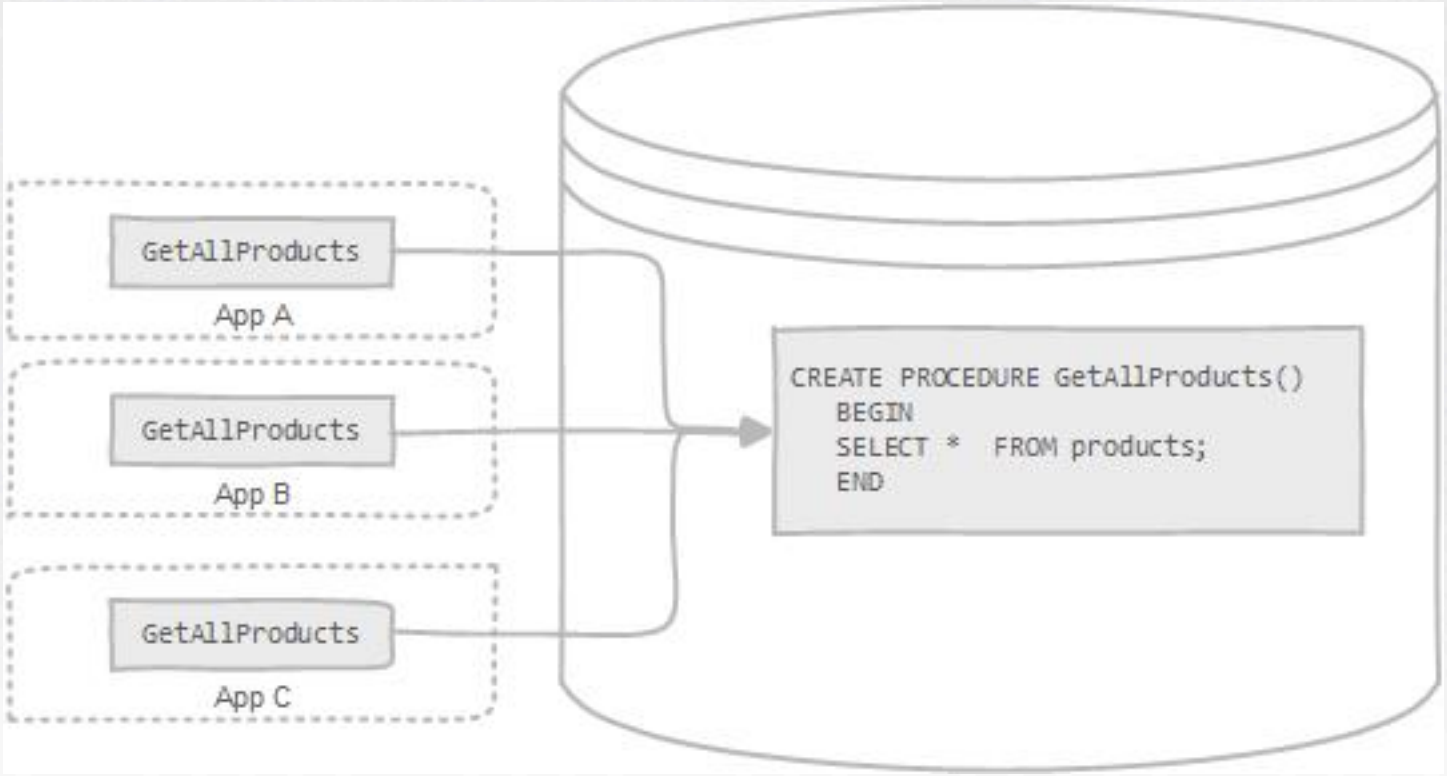
UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- A **procedure** (often called a stored procedure) is a subroutine like a subprogram in a regular computing language, stored in database.
- A procedure has a **name**, a **parameter list**, and **SQL statement(s)**.
- MySQL supports two kinds of "routines": **stored procedures** which you call, or **functions** whose return values you use in other SQL statements the same way that you use pre-installed MySQL functions like pi().
- The major difference is that UDFs can be used like any other expression within SQL statements, whereas stored procedures must be invoked using the CALL statement.

Stored procedures

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Can be invoked by triggers, other stored procedures, and applications such as Java, Python, PHP, etc.



Stored procedures

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Advantages
  - Increase the performance of the application
  - Reduce traffic between application and db server
  - Reusable and transparent to any applications
  - Secure
- Disadvantages
  - Memory usage is not efficient
  - Stored procedure's constructs are not designed for developing complex and flexible business logic
  - Difficult to debug
  - Not easy to develop and maintain

Writing the first MySQL stored procedure

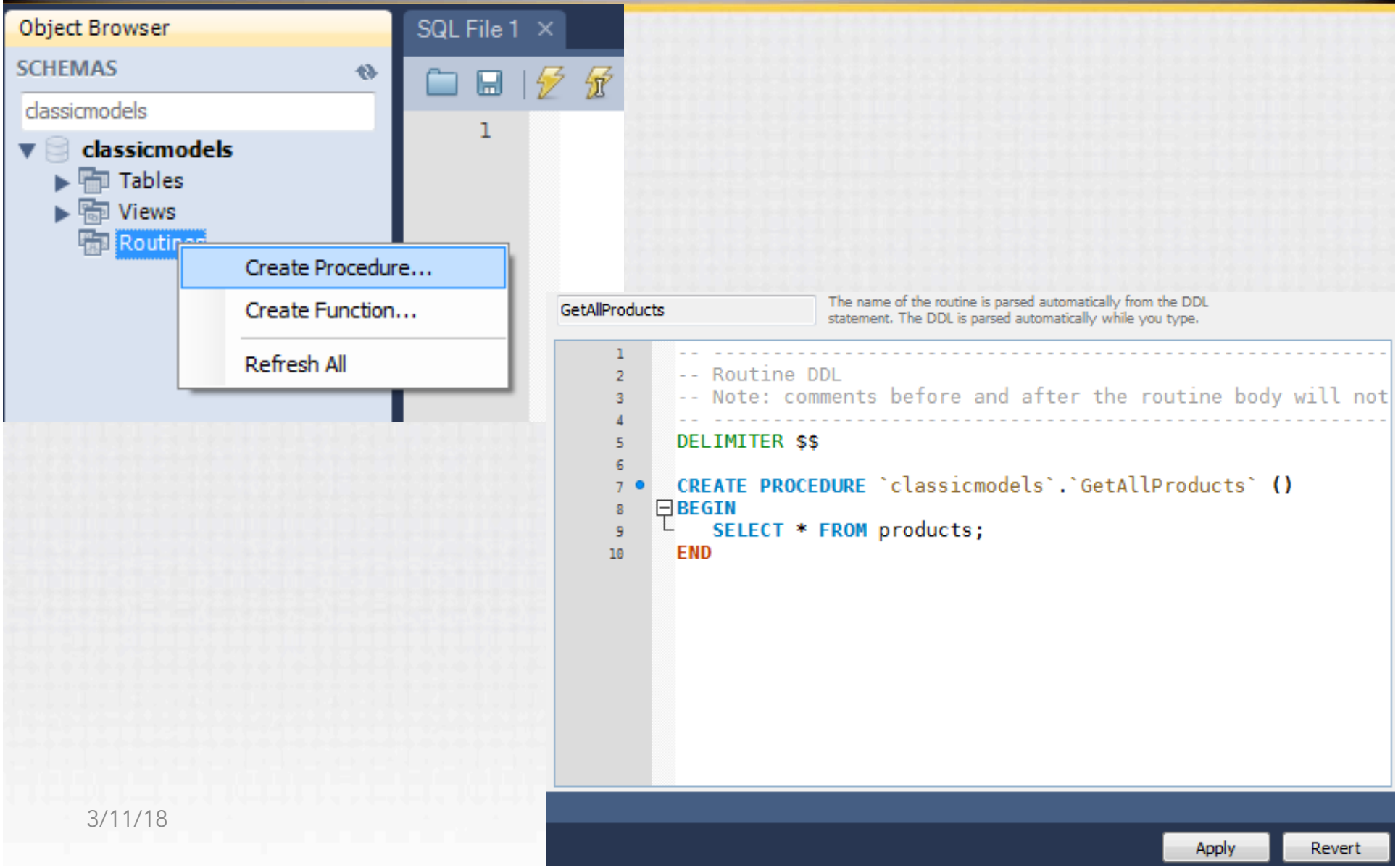
UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
1  DELIMITER //
2  CREATE PROCEDURE GetAllProducts()
3    BEGIN
4    SELECT *  FROM products;
5    END //
6  DELIMITER ;
```

```
mysql> use classicmodels;
Database changed
mysql> DELIMITER //
mysql> CREATE PROCEDURE GetAllProducts()
    -> BEGIN
    -> SELECT * FROM products;
    -> END//
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql>
```

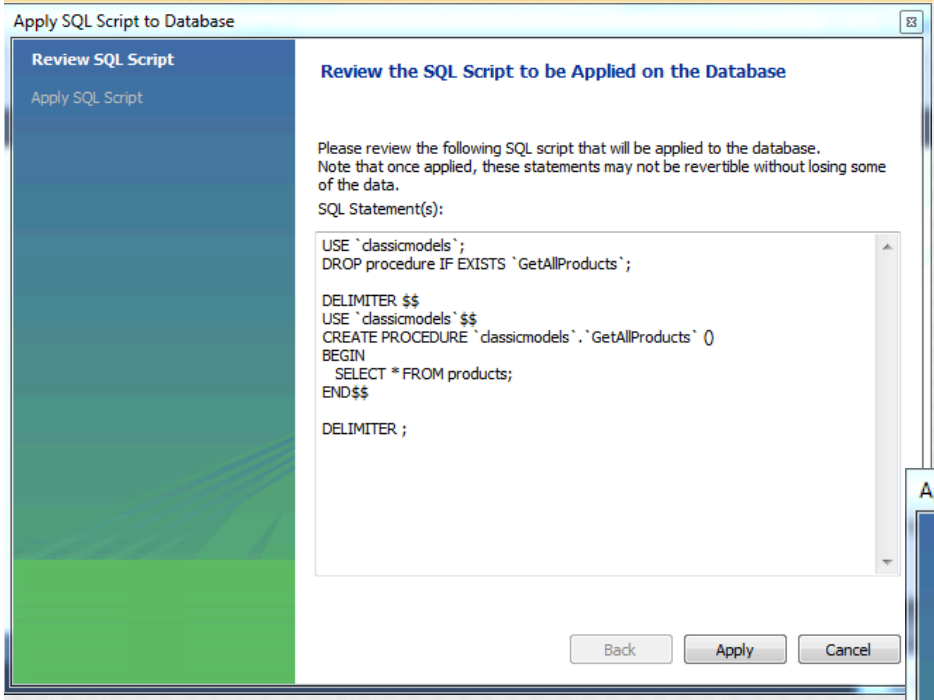
Writing the first MySQL stored procedure

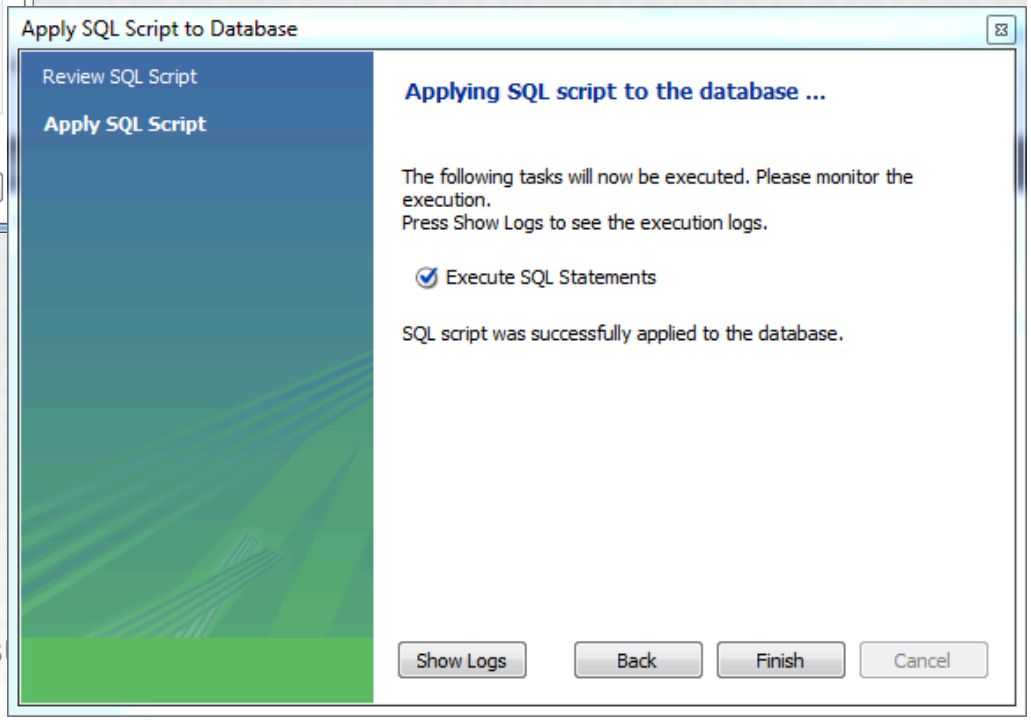
UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS



Writing the first MySQL stored procedure

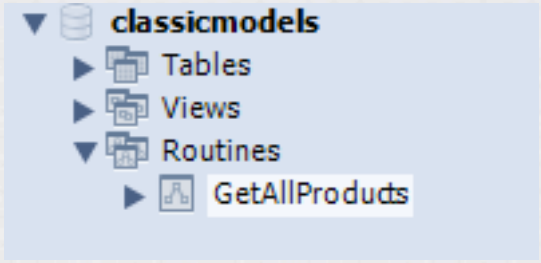
UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS





Writing the first MySQL stored procedure

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS




```
1  CALL GetAllProducts();
```

Filter: File: Autosize:

productCode	productName	productLine	productScale
S10_1678	1969 Harley Davidson Ultimate Chopper	Motorcycles	1:10
S10_1949	1952 Alpine Renault 1300	Classic Cars	1:10
S10_2016	1996 Moto Guzzi 1100i	Motorcycles	1:10
S10_4698	2003 Harley-Davidson Eagle Drag Bike	Motorcycles	1:10
S10_4757	1972 Alfa Romeo GTA	Classic Cars	1:10

Stored procedure in MySQL

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Stored procedures are declared using the following syntax:

```
create procedure <proc-name>
    (param_spec1, param_spec2, …,
   param_specn ) (param_size)
begin
   -- execution code
end;
```

where each param_spec is of the form:

[in | out | inout] <param_name> <param_type>

Variables

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Declaring variables

```
DECLARE variable_name datatype(size) DEFAULT default_value;
```

- Assigning variables

```
DECLARE total_count INT DEFAULT 0;
SET total_count = 10;
```

```
DECLARE total_products INT DEFAULT 0

SELECT COUNT(*) INTO total_products
FROM products
```

- Variable scopes
  - A variable has its own scope that defines its lifetime. If you declare a variable inside a stored procedure, it will be out of scope when the END statement of stored procedure reached.

Procedure parameters

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- A parameter has 3 modes: IN (default), OUT, and INOUT

- `IN` – is the default mode. When you define an `IN` parameter in a stored procedure, the calling program has to pass an argument to the stored procedure. In addition, the value of an `IN` parameter is protected. It means that even the value of the `IN` parameter is changed inside the stored procedure, its original value is retained after the stored procedure ends. In other words, the stored procedure only works on the copy of the `IN` parameter.
- `OUT` – the value of an `OUT` parameter can be changed inside the stored procedure and its new value is passed back to the calling program. Notice that the stored procedure cannot access the initial value of the `OUT` parameter when it starts.
- `INOUT` – an `INOUT` parameter is the combination of `IN` and `OUT` parameters. It means that the calling program may pass the argument, and the stored procedure can modify the `INOUT` parameter and pass the new value back to the calling program.

IN parameter example


```
MODE param_name param_type(param_size)
```

```
DELIMITER //
CREATE PROCEDURE GetOfficeByCountry(IN countryName VARCHAR(255))
 BEGIN
 SELECT *
 FROM offices
 WHERE country = countryName;
 END //
DELIMITER ;
```

CALL GetOfficeByCountry('USA')

officeCode	city	phone	addressLine1	addressLine2	state	country	postalCode	territory
1	San Francisco	+1 650 219 4782	100 Market Street	Suite 300	CA	USA	94080	NA
2	Boston	+1 215 837 0825	1550 Court Place	Suite 102	MA	USA	02107	NA
3	NYC	+1 212 555 3000	523 East 53rd Street	apt. 5A	NY	USA	10022	NA

OUT parameter example

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
1  DELIMITER $$
2  CREATE PROCEDURE CountOrderByStatus(
3   IN orderStatus VARCHAR(25),
4   OUT total INT)
5  BEGIN
6   SELECT count(orderNumber)
7   INTO total
8   FROM orders
9   WHERE status = orderStatus;
10 END$$
11 DELIMITER ;
```

CALL CountOrderByStatus('Shipped', @total)
SELECT @total;

	@total
▶	303

CALL CountOrderByStatus('Shipped', @total)
SELECT @total AS total_in_process;

	total_in_process
▶	6

INOUT parameter example

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
1 DELIMITER $$
2 CREATE PROCEDURE set_counter(INOUT count INT(4),IN inc INT(4))
3 BEGIN
4  SET count = count + inc;
5 END$$
6 DELIMITER ;
```

```
1 SET @counter = 1;
2 CALL set_counter(@counter,1);
3 CALL set_counter(@counter,1);
4 CALL set_counter(@counter,5);
```

Return multiple values

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
DELIMITER $$

CREATE PROCEDURE get_order_by_cust(
 IN cust_no INT,
 OUT shipped INT,
 OUT canceled INT,
 OUT resolved INT,
 OUT disputed INT)
BEGIN
 -- shipped
 SELECT
           count(*) INTO shipped
       FROM
           orders
       WHERE
           customerNumber = cust_no
               AND status = 'Shipped';

 -- canceled
 SELECT
           count(*) INTO canceled
       FROM
           orders
       WHERE
           customerNumber = cust_no
               AND status = 'Canceled';

 -- resolved
 SELECT
           count(*) INTO resolved
       FROM
           orders
       WHERE
           customerNumber = cust_no
               AND status = 'Resolved';

 -- disputed
 SELECT
           count(*) INTO disputed
       FROM
           orders
       WHERE
           customerNumber = cust_no
               AND status = 'Disputed';

END
```

```
CALL get_order_by_cust(141,@shipped,@canceled,@resolved,@disputed);
SELECT @shipped,@canceled,@resolved,@disputed;
```

	@shipped	@canceled	@resolved	@disputed
▶	22	0	1	1

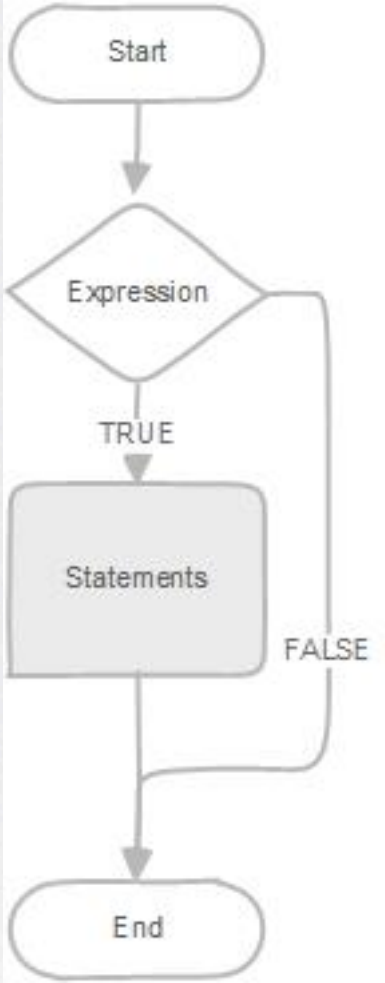
IF

IF <condition> then

<statements>

END IF

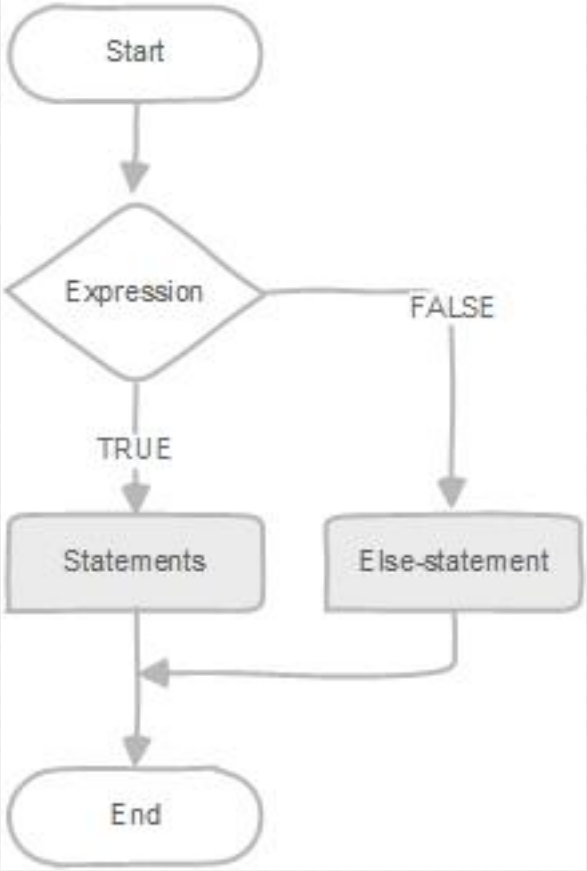
- Note the annoying syntax: END IF has an embedded blank, ELSEIF does not.



IF

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

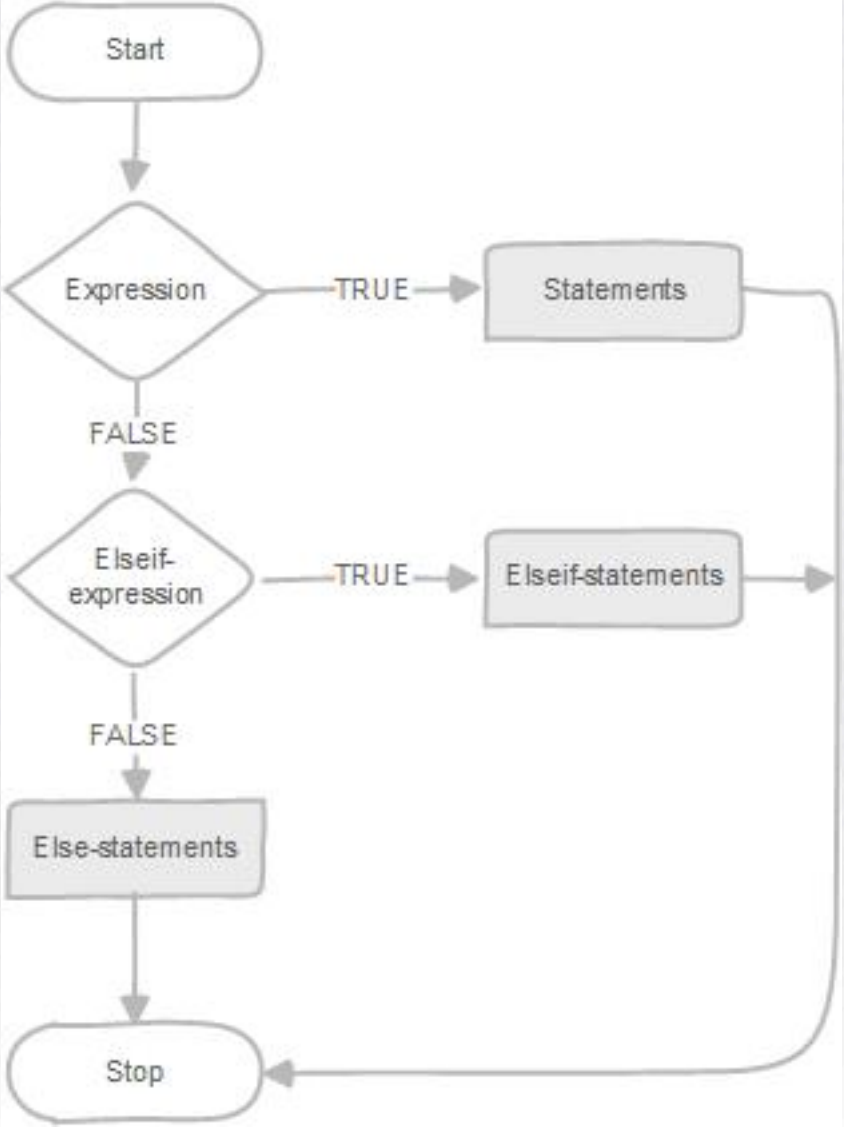
```
IF <condition> then
    <statements>
ELSE
    <statements>
END IF
```



IF

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

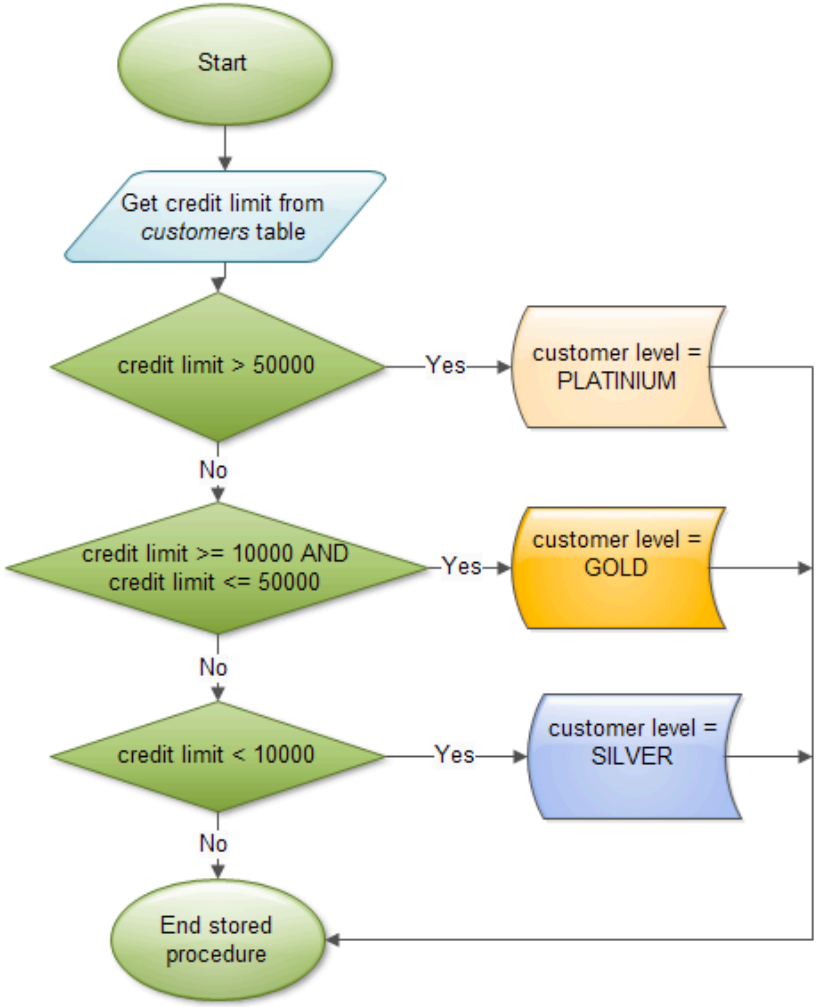
```
IF <condition> then
    <statements>
ELSEIF <condition> then
    <statements>
ELSE
    <statements>
END IF
```



IF example

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
1  DELIMITER $$
2
3  CREATE PROCEDURE GetCustomerLevel(
4      in  p_customerNumber int(11),
5      out p_customerLevel  varchar(10))
6  BEGIN
7      DECLARE creditlim double;
8
9      SELECT creditlimit INTO creditlim
10     FROM customers
11     WHERE customerNumber = p_customerNumber;
12
13     IF creditlim > 50000 THEN
14   SET p_customerLevel = 'PLATINUM';
15     ELSEIF (creditlim <= 50000 AND creditlim >= 10000) THEN
16         SET p_customerLevel = 'GOLD';
17     ELSEIF creditlim < 10000 THEN
18         SET p_customerLevel = 'SILVER';
19     END IF;
20
21 END$$
```



Case statement

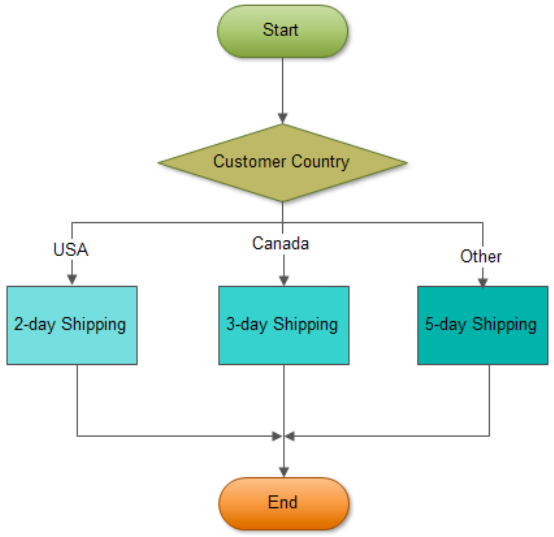
UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
CASE <expression>
    WHEN <value> then
        <statements>
    WHEN <value> then
        <statements>
    …
    ELSE
        <statements>
END CASE;
```

Case statement

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
DELIMITER $$

CREATE PROCEDURE GetCustomerShipping(
 in  p_customerNumber int(11),
 out p_shiping        varchar(50))
BEGIN
    DECLARE customerCountry varchar(50);

    SELECT country INTO customerCountry
 FROM customers
 WHERE customerNumber = p_customerNumber;

    CASE customerCountry
 WHEN  'USA' THEN
    SET p_shiping = '2-day Shipping';
 WHEN 'Canada' THEN
    SET p_shiping = '3-day Shipping';
 ELSE
    SET p_shiping = '5-day Shipping';
 END CASE;

END$$
```




```
SET @customerNo = 112;

SELECT country into @country
FROM customers
WHERE customernumber = @customerNo;

CALL GetCustomerShipping(@customerNo,@shipping);

SELECT @customerNo AS Customer,
       @country    AS Country,
       @shipping   AS Shipping;
```

Customer	Country	Shipping
112	USA	2-day Shipping

Case statement

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
CASE
    WHEN <condition> then
        <statements>
    WHEN <condition> then
        <statements>
    …
    ELSE
        <statements>
END CASE;
```

Case statement

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
DELIMITER $$

CREATE PROCEDURE GetCustomerLevel(
 in  p_customerNumber int(11),
 out p_customerLevel  varchar(10))
BEGIN
    DECLARE creditlim double;

    SELECT creditlimit INTO creditlim
 FROM customers
 WHERE customerNumber = p_customerNumber;

    CASE
 WHEN creditlim > 50000 THEN
    SET p_customerLevel = 'PLATINUM';
 WHEN (creditlim <= 50000 AND creditlim >= 10000) THEN
    SET p_customerLevel = 'GOLD';
 WHEN creditlim < 10000 THEN
    SET p_customerLevel = 'SILVER';
 END CASE;

END$$
```

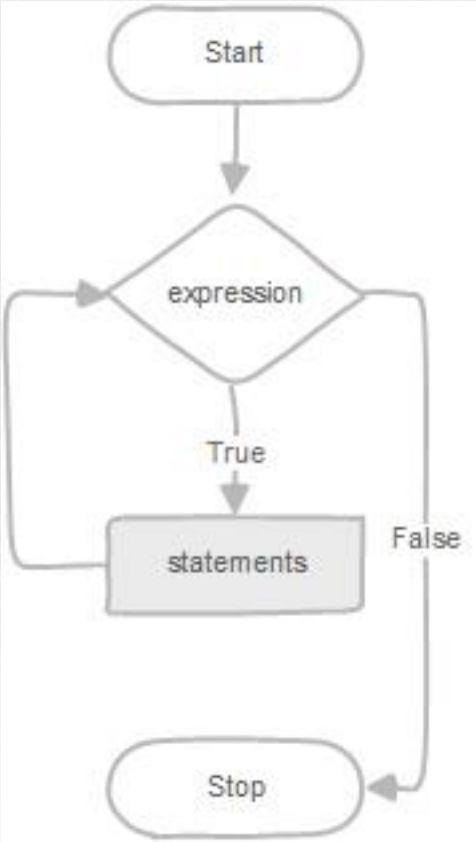
```
CALL GetCustomerLevel(112,@level);
SELECT @level AS 'Customer Level';
```

Customer Level
PLATINUM

Looping

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
WHILE expression DO
   statements
END WHILE
```

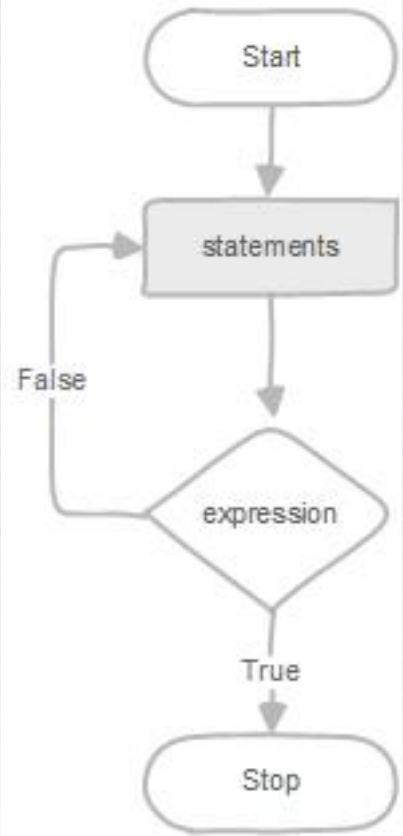



```
DELIMITER $$
DROP PROCEDURE IF EXISTS test_mysql_while_loop$$
CREATE PROCEDURE test_mysql_while_loop()
BEGIN
DECLARE x  INT;
DECLARE str  VARCHAR(255);

SET x = 1;
SET str =  '';

WHILE x  <= 5 DO
SET  str = CONCAT(str,x,',');
SET  x = x + 1;
END WHILE;

SELECT str;
END$$
DELIMITER ;
```

Looping

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
REPEAT
 statements;
UNTIL expression
END REPEAT
```




```
DELIMITER $$
DROP PROCEDURE IF EXISTS mysql_test_repeat_loop$$
CREATE PROCEDURE mysql_test_repeat_loop()
BEGIN
DECLARE x INT;
DECLARE str VARCHAR(255);

SET x = 1;
       SET str =  '';

REPEAT
SET  str = CONCAT(str,x,',');
SET  x = x + 1;
       UNTIL x  > 5
       END REPEAT;

       SELECT str;
END$$
DELIMITER ;
```

Looping

- The LEAVE statement allows you to exit the loop immediately without waiting for checking the condition.
- The ITERATE statement allows you to skip the entire code under it and start a new iteration.

```
CREATE PROCEDURE test_mysql_loop()
 BEGIN
 DECLARE x  INT;
        DECLARE str  VARCHAR(255);

 SET x = 1;
        SET str =  '';

 loop_label:  LOOP
 IF  x > 10 THEN
 LEAVE  loop_label;
 END  IF;

 SET  x = x + 1;
 IF  (x mod 2) THEN
 ITERATE  loop_label;
 ELSE
                SET  str = CONCAT(str,x,',');
 END  IF;
         END LOOP;

         SELECT str;

 END;
```

Stored Procedures in MySQL

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Use show procedure status to display the list of stored procedures you have created

```
SHOW PROCEDURE STATUS [LIKE 'pattern' | WHERE expr];
```

```
mysql> show procedure status;
+------+--------------+-----------+---------+---------------------+---------------------+----------
-----+---------+----------------------+----------------------+--------------------+
| Db   | Name         | Type      | Definer | Modified            | Created             | Security_
type | Comment | character_set_client | collation_connection | Database Collation |
+------+--------------+-----------+---------+---------------------+---------------------+----------
-----+---------+----------------------+----------------------+--------------------+
| ptan | updateSalary0 | PROCEDURE | ptan@%  | 2010-03-16 12:21:55 | 2010-03-16 12:21:55 | DEFINER
     |         | latin1               | latin1_swedish_ci    | latin1_swedish_ci  |
+------+--------------+-----------+---------+---------------------+---------------------+----------
-----+---------+----------------------+----------------------+--------------------+
1 row in set (0.02 sec)
```

- Display stored procedure's source code

```
SHOW CREATE PROCEDURE stored_procedure_name
```

- Use drop procedure to remove a stored procedure

```
mysql> drop procedure updateSalary;
Query OK, 0 rows affected (0.00 sec)
```

Functions

- They take arguments and return a single output.
- The general syntax is: create function <name> (<arg1> <type1>, [<arg2> <type2> [,…]) returns <return type> [deterministic]
  - Deterministic means that the output from the function is strictly a consequence of the arguments.
  - Same values input → same values output.
  - Note that the arguments cannot be changed and the new values passed back to the caller.
- Follow that with begin … end and you have a function.

Functions

function <function-name> (param_spec$_1$, …, param_spec$_k$)
    returns <return_type>
    [deterministic] allow optimization if same output
                for the same input (use RAND not deterministic)
Begin
  -- execution code
end;
where param_spec is:
    [in | out | in out] <param_name> <param_type>

Example of Functions

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
mysql> select * from employee;
+----+------+---------+--------+------------+------+
| id | name | superid | salary | bdate      | dno  |
+----+------+---------+--------+------------+------+
|  1 | john |       3 | 100000 | 1960-01-01 |    1 |
|  2 | mary |       3 |  50000 | 1964-12-01 |    3 |
|  3 | bob  |    NULL |  80000 | 1974-02-07 |    3 |
|  4 | tom  |       1 |  50000 | 1970-01-17 |    2 |
|  5 | bill |    NULL |   NULL | 1985-01-20 |    1 |
+----+------+---------+--------+------------+------+
5 rows in set (0.00 sec)

mysql> delimiter |
mysql> create function giveRaise (oldval double, amount double
    -> returns double
    -> deterministic
    -> begin
    ->       declare newval double;
    ->       set newval = oldval * (1 + amount);
    ->       return newval;
    -> end |
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
```

Another Example of Functions

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
mysql> select name, salary, giveRaise(salary, 0.1) as newsal
    -> from employee;
+------+--------+--------+
| name | salary | newsal |
+------+--------+--------+
| john | 100000 | 110000 |
| mary |  50000 |  55000 |
| bob  |  80000 |  88000 |
| tom  |  50000 |  55000 |
| bill |   NULL |   NULL |
+------+--------+--------+
5 rows in set (0.00 sec)
```

SQL Triggers

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- To monitor a database and take a corrective action when a condition occurs
  - Examples:
    - Charge $10 overdraft fee if the balance of an account after a withdrawal transaction is less than $500
    - Limit the salary increase of an employee to no more than 5% raise

CREATE TRIGGER trigger-name
trigger-time trigger-event
ON table-name
FOR EACH ROW
    trigger-action;

trigger-time ∈ {BEFORE, AFTER}

trigger-event ∈ {INSERT,DELETE,UPDATE}

Triggers

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Please see: http://dev.mysql.com/doc/refman/5.7/en/create-trigger.html for the complete specification for triggers.

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    TRIGGER trigger_name
    trigger_time    trigger_event
    ON tbl_name FOR EACH ROW
    [trigger_order]
    trigger_body

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

trigger_order: { FOLLOWS | PRECEDES } other_trigger_name
```

SQL Triggers: Example

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- We want to create a trigger to update the total salary of a department when a new employee is hired

```
mysql> select * from employee;
+----+------+---------+--------+------------+------+
| id | name | superid | salary | bdate      | dno  |
+----+------+---------+--------+------------+------+
|  1 | john |       3 | 100000 | 1960-01-01 |    1 |
|  2 | mary |       3 |  50000 | 1964-12-01 |    3 |
|  3 | bob  |    NULL |  80000 | 1974-02-07 |    3 |
|  4 | tom  |       1 |  50000 | 1970-01-17 |    2 |
|  5 | bill |    NULL |   NULL | 1985-01-20 |    1 |
+----+------+---------+--------+------------+------+
5 rows in set (0.00 sec)

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      100000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)
```

SQL Triggers: Example

- Create a trigger to update the total salary of a department when a new employee is hired:

```
mysql> delimiter |
mysql> create trigger update_salary
    -> after insert on employee
    -> for each row
    -> begin
    ->       if new.dno is not null then
    ->           update deptsal
    ->           set totalsalary = totalsalary + new.salary
    ->           where dnumber = new.dno;
    ->       end if;
    -> end |
Query OK, 0 rows affected (0.06 sec)

mysql> delimiter ;
```

- The keyword "new" refers to the new row inserted

SQL Triggers: Example – Part 2


```
mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      100000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)

mysql> insert into employee values (6,'lucy',null,90000,'1981-01-01',1);
Query OK, 1 row affected (0.08 sec)

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      190000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)

mysql> insert into employee values (7,'george',null,45000,'1971-11-11',null);
Query OK, 1 row affected (0.02 sec)

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      190000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)

mysql> drop trigger update_salary;
Query OK, 0 rows affected (0.00 sec)
```

totalsalary increases by 90K

totalsalary did not change

SQL Triggers: Example – Part 3

- A trigger to update the total salary of a department when an employee tuple is modified:

```
mysql> delimiter !
mysql> create trigger update_salary2
    -> after update on employee
    -> for each row
    -> begin
    ->       if old.dno is not null then
    ->          update deptsal
    ->          set totalsalary = totalsalary - old.salary
    ->          where dnumber = old.dno;
    ->       end if;
    ->       if new.dno is not null then
    ->          update deptsal
    ->          set totalsalary = totalsalary + new.salary
    ->          where dnumber = new.dno;
    ->       end if;
    -> end !
Query OK, 0 rows affected (0.06 sec)
```

SQL Triggers: Example – Part 4

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
mysql> deliniter ;
mysql> select * from employee;
+----+--------+---------+--------+------------+------+
| id | name   | superid | salary | bdate      | dno  |
+----+--------+---------+--------+------------+------+
|  1 | john   |       3 | 100000 | 1960-01-01 |    1 |
|  2 | mary   |       3 |  50000 | 1964-12-01 |    3 |
|  3 | bob    |    NULL |  80000 | 1974-02-07 |    3 |
|  4 | tom    |       1 |  50000 | 1970-01-17 |    2 |
|  5 | bill   |    NULL |   NULL | 1985-01-20 |    1 |
|  6 | lucy   |    NULL |  90000 | 1981-01-01 |    1 |
|  7 | george |    NULL |  45000 | 1971-11-11 | NULL |
+----+--------+---------+--------+------------+------+
7 rows in set (0.00 sec)

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      190000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)

mysql> update employee set salary = 100000 where id = 6;
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      200000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)
```

SQL Triggers: Example – Part 5

- A trigger to update the total salary of a department when an employee tuple is deleted:

```
mysql> delimiter !
mysql> create trigger update_salary3
    -> before delete on employee
    -> for each row
    -> begin
    ->       if (old.dno is not null) then
    ->           update deptsal
    ->           set totalsalary = totalsalary - old.salary
    ->           where dnumber = old.dno;
    ->       end if;
    -> end !
Query OK, 0 rows affected (0.08 sec)

mysql> delimiter ;
```

SQL Triggers: Example – Part 6

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
mysql> select * from employee;
+----+--------+---------+--------+------------+------+
| id | name   | superid | salary | bdate      | dno  |
+----+--------+---------+--------+------------+------+
|  1 | john   |       3 | 100000 | 1960-01-01 |    1 |
|  2 | mary   |       3 |  50000 | 1964-12-01 |    3 |
|  3 | bob    |    NULL |  80000 | 1974-02-07 |    3 |
|  4 | tom    |       1 |  50000 | 1970-01-17 |    2 |
|  5 | bill   |    NULL |   NULL | 1985-01-20 |    1 |
|  6 | lucy   |    NULL | 100000 | 1981-01-01 |    1 |
|  7 | george |    NULL |  45000 | 1971-11-11 | NULL |
+----+--------+---------+--------+------------+------+
7 rows in set (0.00 sec)
```

```
mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      200000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)
```

```
mysql> delete from employee where id = 6;
Query OK, 1 row affected (0.02 sec)

mysql> delete from employee where id = 7;
Query OK, 1 row affected (0.03 sec)

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      100000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)
```

A Few Things to Note

- A given trigger can only have one event.
- If you have the same or similar processing that has to go on during insert and delete, then it's best to have that in a procedure or function and then call it from the trigger.
- A good naming standard for a trigger is **<table_name>_event** if you have the room for that in the name.
- Just like a function or a procedure, the trigger body will need a begin … end unless it is a single statement trigger.

The Special Powers of a Trigger

- While in the body of a trigger, there are potentially two sets of column values available to you, with special syntax for denoting them.
  - old.<column name> will give you the value of the column **before** the DML statement executed.
  - new.<column name> will give you the value of that column **after** the DML statement executed.
- Insert triggers have no old values available, and delete triggers have no new values available for obvious reasons. Only update triggers have both the old and the new values available.
- Only triggers can access these values this way.

More Examples

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Simplified example of a parent table: hospital_room as the parent and hospital_bed as the child.
- The room has a column: *max_beds* that dictates the maximum number of beds for that room and a column: *hospital_room_no*.
- The hospital_bed table has column *hospital_bed_id* and *room_id*.
- The hospital_bed table has a before insert trigger that checks to make sure that the hospital room does not already have its allotted number of beds.

The Trigger

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
CREATE DEFINER=`root`@`localhost`
TRIGGER `hospital`.`hospital_bed_BEFORE_INSERT`
BEFORE INSERT ON `hospital_bed` FOR EACH ROW
BEGIN
	declare  max_beds_per_room int;
	declare  current_count int;

	select	max_beds into max_beds_per_room
	from	hospital_room
	where	hospital_room_no = new.room_id;

	select	count(*) into current_count
	from	hospital_bed
	where	room_id = new.room_id;

	if current_count >= max_beds_per_room then
		signal sqlstate '45000' set message_text='Too many beds in that room already!';
	end if;
END;
```

Firing the trigger


```
insert    into hospital_bed (room_id, hospital_bed_id) values ('323B', 1);
insert    into hospital_bed (room_id, hospital_bed_id) values ('323B', 2);
insert    into hospital_bed (room_id, hospital_bed_id) values ('323B', 3);
insert    into hospital_bed (room_id, hospital_bed_id) values ('323B', 4);
insert    into hospital_bed (room_id, hospital_bed_id) values ('323B', 5);
```

Error Code: 1644. Too many beds in that room already!

Using a stored procedure instead

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `hospital`.`too_many_beds`(in room_id varchar(45))
BEGIN
	declare max_beds_per_room int;
	declare current_count int;
	declare room_count int;

	-- see if the hospital room exists
	select  count(*) into room_count from hospital_room where  hospital_room_no = room_id;
	if room_count = 1 then -- we can see if room for 1 more bed
	begin
		select  max_beds into max_beds_per_room from hospital_room where  hospital_room_no=room_id;
		-- count the beds in this room

		select  count(*) into current_count from hospital_bed where  room_id = room_id;

		if current_count >= max_beds_per_room then
			-- flag an error to abort if necessary
			signal sqlstate '45000' set message_text='Too many beds in that room already!';
		end if;
	end;
	end if;
END
```

Comments on the procedure

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Because that is in isolation from the beds table, we have to check to make sure that the room number is viable.

- As a stored procedure, this can be called directly from the command line as a means of unit testing.

Viewing your triggers

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- MySQL has a schema that has tables for all of the information that is needed to define and run the data in the database. This is meta data.
- select * from information_schema.triggers where trigger_schema='<your schema name>';
  - retrieve the trigger information for the triggers in <your schema name>.
- Alternatively, you can use the "show triggers" command (this is not SQL) that will display a report of your triggers from the default schema.

  mysql> show triggers;