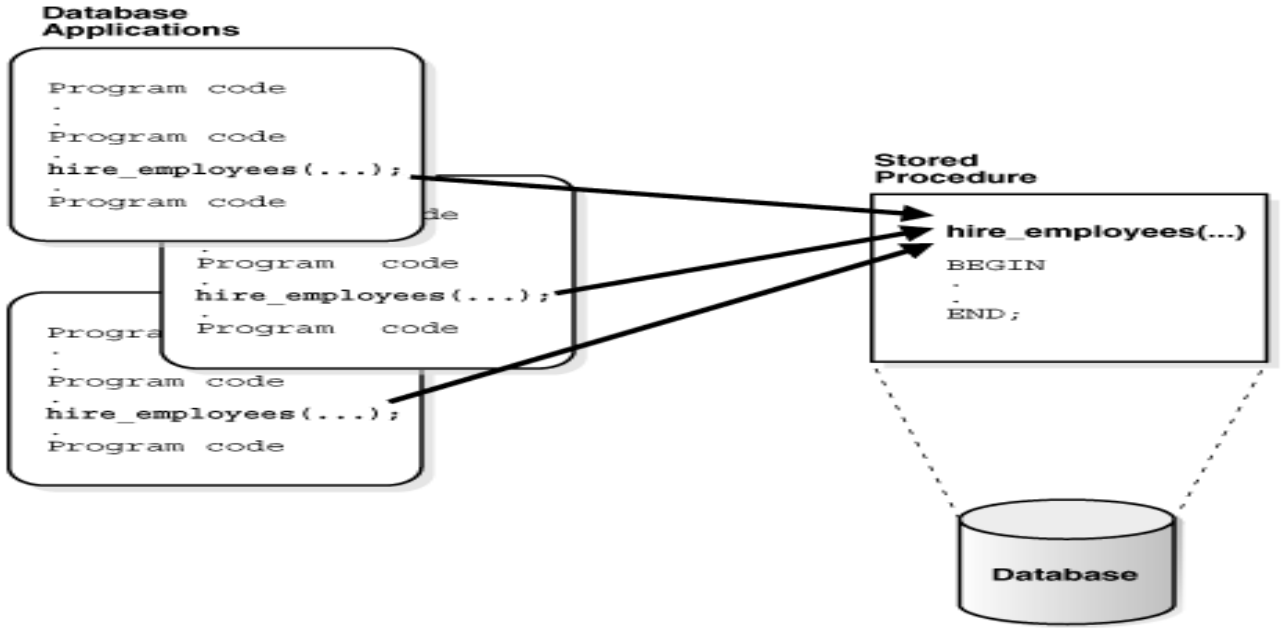
CPSC 5021: Database Systems

Stored Procedure

Lin Li

Stored Procedures

- A stored procedure is a set of SQL statements that can be stored in the server.
- With stored procedures, clients don't need to keep reissuing the individual statements. Instead, just simply call the stored procedure.



Create Stored Routines

```
CREATE
  PROCEDURE sp_name ([proc_parameter[,...]])
  routine_body
```

```
CREATE
  FUNCTION sp_name ([func_parameter[,...]])
  RETURNS type
  routine_body
```

type:
Any valid MySQL data type

routine_body
Valid MySQL statement

proc_parameter:
[IN | OUT | INOUT] param_name type

func_parameter:
param_name type

Variables

- Declare variables:

```
DECLARE var_name[,...] type [DEFAULT value]
```

- Assign values:

```
SET var_name = expr [, var_name = expr] ...
SELECT col_name[,...] INTO var_name[,...] FROM table_expr
```

- Example:

```
DELIMITER |
CREATE PROCEDURE testVariable(INOUT totalcount INT)
BEGIN
  DECLARE count INT DEFAULT 0;
  SELECT COUNT(*) INTO count FROM PRODUCT;
  SET totalcount = count + totalcount;
END
|
```

```
SET @totalcount = 10;
CALL testVariable(@totalcount);
SELECT @totalcount;
```

Parameters

- Each parameter is an IN parameter by default. You can also explicitly specify a parameter to be OUT or INOUT parameter if you define a PROCEDURE(not FUNCTION).

  - An IN parameter passes a value into a procedure. The procedure might modify the value, but the modification is not visible to the caller when the procedure returns.

  - An OUT parameter passes a value from the procedure back to the caller. Its initial value is NULL within the procedure, and its value is visible to the caller when the procedure returns.

  - An INOUT parameter is initialized by the caller. It can be modified by the procedure, and any change made by the procedure is visible to the caller when the procedure returns.

- Specifying a parameter as IN, OUT or INOUT is valid only for a PROCEDURE. For FUNCTION, parameters are all IN.

IN Example

```
delimiter |
CREATE PROCEDURE changeCourse(code varchar(6), name varchar(70))
BEGIN
   UPDATE COURSES
   SET COURSE_NAME = name
   WHERE COURSE_CODE = code;
   SELECT * FROM COURSES
   WHERE COURSE_CODE = code;
END
|

CALL changeCourse('CS056', 'Data Structure');
```

OUT Example

```
delimiter |
CREATE PROCEDURE stuRegistration(IN stuID CHAR(11), OUT count INT)
BEGIN
  SELECT COUNT(*) INTO count FROM REGISTRATION
  WHERE STUDENT_ID = stuID;
END
|
```

```
SET @ID = '861103-2438';
CALL stuRegistration(@ID, @numberofCourses);
SELECT @ID, @numberofCourses;
```

INOUT Example

```
delimiter |
CREATE PROCEDURE swap(INOUT param1 INT, INOUT param2 INT)
BEGIN
  # swap two numbers
  DECLARE temp INT;
  SET temp = param1;
  SET param1 = param2;
  SET param2 = temp;
END
|
```

```
SET @number1 = 10, @number2 = 12;
CALL swap(@number1, @number2);
SELECT @number1, @number2;
```

Function Returns

- It is mandatory that a FUNCTION includes the RETURNS clause, which indicates the return type of the function.
- The function body must contain a RETURN value statement.

```
CREATE FUNCTION hello (param CHAR(20))
RETURNS CHAR(50)
RETURN CONCAT('Hello, ', param, '!');

SELECT hello('world');
```

Modify/Delete a Stored Routine

- Alter Procedure/Function: If you need to alter the body or the parameters, you must drop and recreate the Procedure/Function.
- Drop procedure:

```
DROP PROCEDURE swap;
```

- Drop function:

```
DROP FUNCTION hello;
```

Exercise

- Question 1 in the hand-out

Control Statements - IF

- Syntax

```
IF if_expression
THEN commands
[ELSEIF elseif_expression THEN commands]
[ELSE commands]
END IF;
```

Control Statements - IF

- Example

```
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;
   IF creditlim > 50000 THEN
      SET p_customerLevel = 'PLATINUM';
   ELSEIF (creditlim <= 50000 AND creditlim >= 10000) THEN
      SET p_customerLevel = 'GOLD';
   ELSEIF creditlim < 10000 THEN
      SET p_customerLevel = 'SILVER';
   END IF;
END
|
```

```
SET @customer = 2;

CALL
getCustomerLevel(@customer,
@customerlevel);

SELECT @customer,
@customerlevel;
```

Control Statements - CASE

- Syntax

```
CASE  case_expression
  WHEN when_expression_1 THEN commands
  WHEN when_expression_2 THEN commands
  ...
  ELSE commands
END CASE;
```

Control Statement - CASE

- Example

```
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 @customer = 2;

CALL getCustomerShipping(@customer,
@customerShipping);

SELECT @customer,
@customerShipping;
```

Exercise

- Question 2 in the hand-out

Control Statements – WHILE Loop

- Syntax

```
WHILE expression DO
  Statements
END WHILE
```

- Example

```
delimiter |
CREATE PROCEDURE whileLoopProc(OUT result varchar(255))
BEGIN
        DECLARE x  INT;
        SET x = 1;
        SET result =  '';
        WHILE x  <= 5 DO
           SET  result = CONCAT(result, x, ',');
           SET  x = x + 1;
        END WHILE;
END
|
```

```
CALL whileLoopProc(@result);

SELECT @result;
```

Control Statements – REPEAT Loop

- Syntax

```
REPEAT
   Statements;
UNTIL expression
END REPEAT
```

- Example

```
DELIMITER |
CREATE PROCEDURE repeatLoopProc(OUT result varchar(255))
BEGIN
      DECLARE x  INT;
      SET x = 1;
      SET result =  '';
      REPEAT
           SET  result = CONCAT(result, x, ',');
           SET  x = x + 1;
      UNTIL x  > 5
      END REPEAT;
END
|
```

```
CALL
repeatLoopProc(@result);

SELECT @result;
```

Control Statements – Leave and Iterate

- The LEAVE statement allows you to exit the loop immediately
- The ITERATE statement allows you to skip the entire code after it in the current iteration and start a new iteration

Control Statements – Leave and Iterate

- Example

```
DELIMITER |
CREATE PROCEDURE loopProc(out result varchar(255))
BEGIN
    DECLARE x INT;
    SET x = 1;
    SET result = '';
    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 result = CONCAT(result, x, ',');
        END IF;
    END LOOP;
END
|
```

```
CALL loopProc(@result);

SELECT @result;
```

Exercise

- Question 3 & 4 in the hand-out