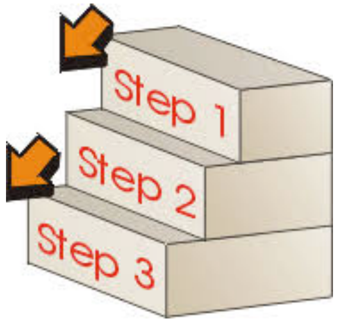
UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

# Information Storage and Management I

Dr. Alejandro Arbelaez



Stored Procedures

## So far…

◒ Integrity Constraints/Triggers

◒ Views/Stored Procedures

● Normal Forms

● Entity Relationship Modelling

● Relational Algebra

Object Relational DBs
Physical Storage
Transactions (Int.)
Indexing

## Creating Stored Procedures

```
DELIMITER //
CREATE PROCEDURE NAME
  BEGIN
        SQL STATEMENT
  END //
DELIMITER ;
```

```
DELIMITER //
CREATE PROCEDURE GetAllProducts()
  BEGIN
  SELECT *  FROM products;
  END //
DELIMITER ;
```

## Calling Stored Procedures

```
CALL GetAllProducts();
```

## Variables

- A variable is a name that refers to a value
- Python:

```
name = “Alex”
age = 35
```

- MySQL

```
DECLARE name VARCHAR(225)
DECLARE age INT
```

## Define parameters within a stored procedure

- Parameter list is empty
  - CREATE PROCEDURE proc1 () :
- Define input parameter with key word IN:
  - CREATE PROCEDURE proc1 (IN varname DATA-TYPE)
  - The word IN is optional because parameters are IN (input) by default.
- Define output parameter with OUT:
  - CREATE PROCEDURE proc1 (OUT varname DATA-TYPE)
- A procedure may have input and output paramters:
  - CREATE PROCEDURE proc1 (INOUT varname DATA-TYPE)

## Three Types of Parameters

- **IN**
  - **Default**
- OUT
- INOUT

## In Parameter

- Calling program has to pass an argument to the stored procedure.

# Arguments and Parameters

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

Defining

```
CALL GetOfficeByCountry('USA')
```

$\Rightarrow$ Calling

## Three Types of Parameters

- IN
  - Default
- **OUT**
- INOUT

## Out 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
- **OUT** is a keyword

## Out Parameter

```
 DELIMITER //
CREATE PROCEDURE CountOrderByStatus(IN orderStatus VARCHAR(25), OUT total INT)
BEGIN
 SELECT count(orderNumber) INTO total FROM orders WHERE status = orderStatus;
END//
DELIMITER ;
```

Defining

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

SELECT @total;
```

The out parameter is used outside of the stored procedure.

## User-Defined Temporary Variables

User variables are written as @var_name.

```
mysql> SET @t1=1, @t2=2, @t3:=4;
mysql> SELECT @t1, @t2, @t3, @t4 := @t1+@t2+@t3;
+------+------+------+--------------------+
| @t1  | @t2  | @t3  | @t4 := @t1+@t2+@t3 |
+------+------+------+--------------------+
|    1 |    2 |    4 |                  7 |
+------+------+------+--------------------+
```

## Example of running the procedure from the command prompt

```
mysql> delimiter ;
mysql> set @tax=0;
Query OK, 0 rows affected (0.00 sec)

mysql> call caltax('S1',0.1,@tax);
Query OK, 1 row affected (0.00 sec)

mysql> select @tax;
+------+
| @tax |
+------+
|  650 |
+------+
1 row in set (0.00 sec)
```

## Three Types of Parameters

- IN
  - Default

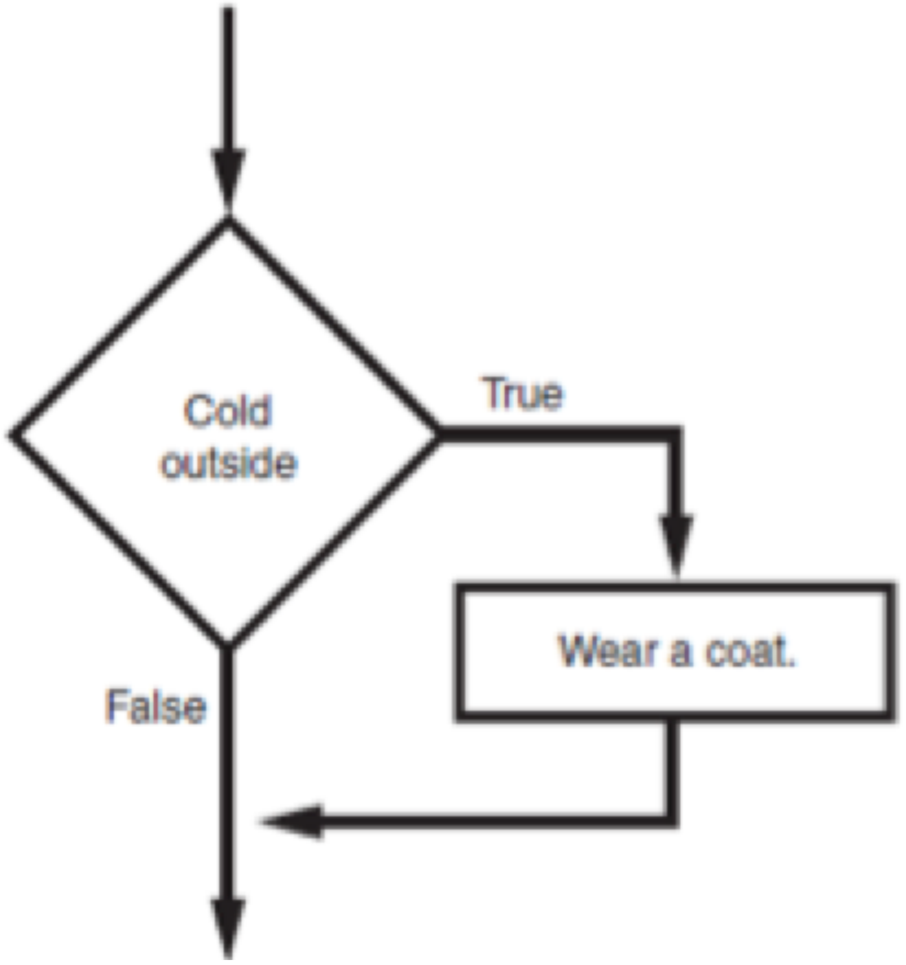
- OUT

- INOUT

## Examples of parameters

```
CREATE PROCEDURE proc_IN (IN var1 INT)
BEGIN
   SELECT var1 + 2 AS result;
END

CREATE PROCEDURE proc_OUT(OUT var1 VARCHAR(100))
BEGIN
 SET var1 = 'This is a test';
END

CREATE PROCEDURE proc_INOUT (IN var1 INT,OUT var2 INT)
BEGIN
   SET var2 = var1 * 2;
END
```

## Conditionals



## The “If” Statement (MySQL Syntax)

IF **if_expression** THEN commands

  [ELSEIF **elseif_expression** THEN commands]

  [ELSE commands]

END IF;

## MySQL Comparison Operators

- EQUAL(=)
- LESS THAN(<)
- LESS THAN OR EQUAL(<=)
- GREATER THAN(>)
- GREATER THAN OR EQUAL(>=)
- NOT EQUAL(<>,!=)

## IF Statement

```
DELIMITER //
CREATE PROCEDURE GetProductsInStockBasedOnQuantitityLevel(IN
p_operator VARCHAR(255), IN p_quantityInStock INT)
 BEGIN
  IF p_operator = "<" THEN
            select * from products WHERE quantityInStock < p_quantityInStock;
  ELSEIF p_operator = ">" THEN
            select * from products WHERE quantityInStock > p_quantityInStock;
  END IF;
 END //
DELIMITER ;
```

## IF Statement

```
CREATE PROCEDURE GetProductsInStockBasedOnQuantitityLevel
(IN p_operator VARCHAR(255), IN p_quantityInStock INT)
```

The operator > or <

The number in stock

## The IF Statement

```
IF p_operator = "<" THEN
        select * from products WHERE quantityInStock <p_quantityInStock;

ELSEIF p_operator = ">" THEN
        select * from products WHERE quantityInStock > p_quantityInStock;
END IF;
```

## Loops

- While
- Repeat
- Loop

Repeats a set of commands until some conditions is met

Iteration: one execution of the body of a loop

If a condition is never met, we will have a infinite loop

## While Loop

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

The expression must evaluate to true or false

while loop is known as a *pretest* loop

Tests condition before performing an iteration

    Will never execute if condition is false to start with

    Requires performing some steps prior to the loop

## Infinite Loops

- Loops must contain within themselves a way to terminate
    - Something inside a while loop must eventually make the condition false
- Infinite loop: loop that does not have a way of stopping
    - Repeats until program is interrupted
    - Occurs when programmer forgets to include stopping code in the loop

## While Loop

```
DELIMITER //
CREATE PROCEDURE WhileLoopProc()
      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 ;
```

## While Loop

Creating Variables

```
DECLARE x  INT;
DECLARE str  VARCHAR(255);
 SET x = 1;
 SET str =  '';
```

## While Loop

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

## Cursors

- A cursor is a pointer to a set of records returned by a SQL statement. It enables you to take a set of records and deal with it on a row-by-row basis.
- To handle a result set inside a stored procedure, you use a cursor. A cursor allows you to iterate a set of rows returned by a query and process each row individually.

## Cursor has three important properties

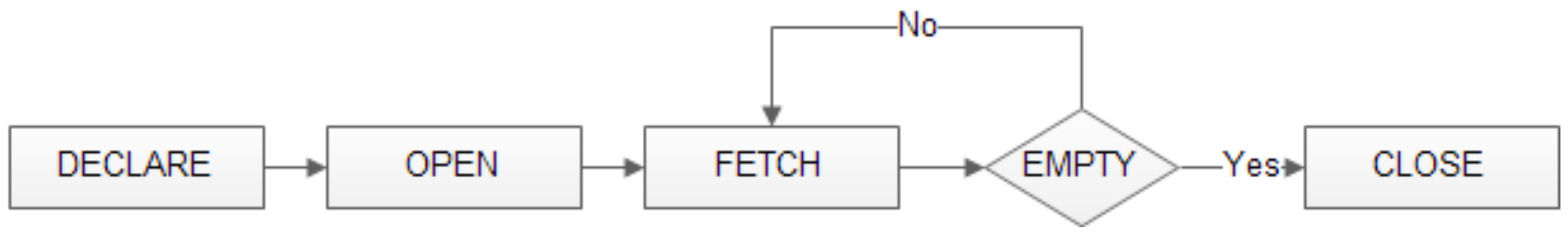
- The cursor will not reflect changes in its source tables.
- Read Only : Cursors are not updatable.
- Not Scrollable : Cursors can be traversed only in one direction, forward, and you can't skip records from fetching.

## Defining and Using Cursors

- Declare cursor:
  - DECLARE cursor-name CURSOR FOR SELECT ...;
- DECLARE  CONTINUE HANDLER FOR NOT FOUND: Specify what to do when no more records found
  - DECLARE  b INT;
  - DECLARE CONTINUE HANDLER FOR NOT FOUND SET b = 1;
- Open cursor:
  - OPEN cursor-name;
- Fetch data into variables:
  - FETCH cursor-name INTO variable [, variable];
- CLOSE cursor:
  - CLOSE cursor-name;

Use the FETCH statement to retrieve the next row pointed by the cursor and move the cursor to the next row in the result set.

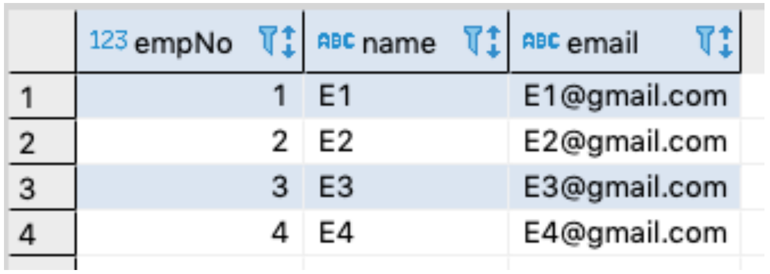
## MySQL Cursor



## A procedure to create email list using cursor

Concatenate all emails where each email is separated by a semicolon(;):

Employee

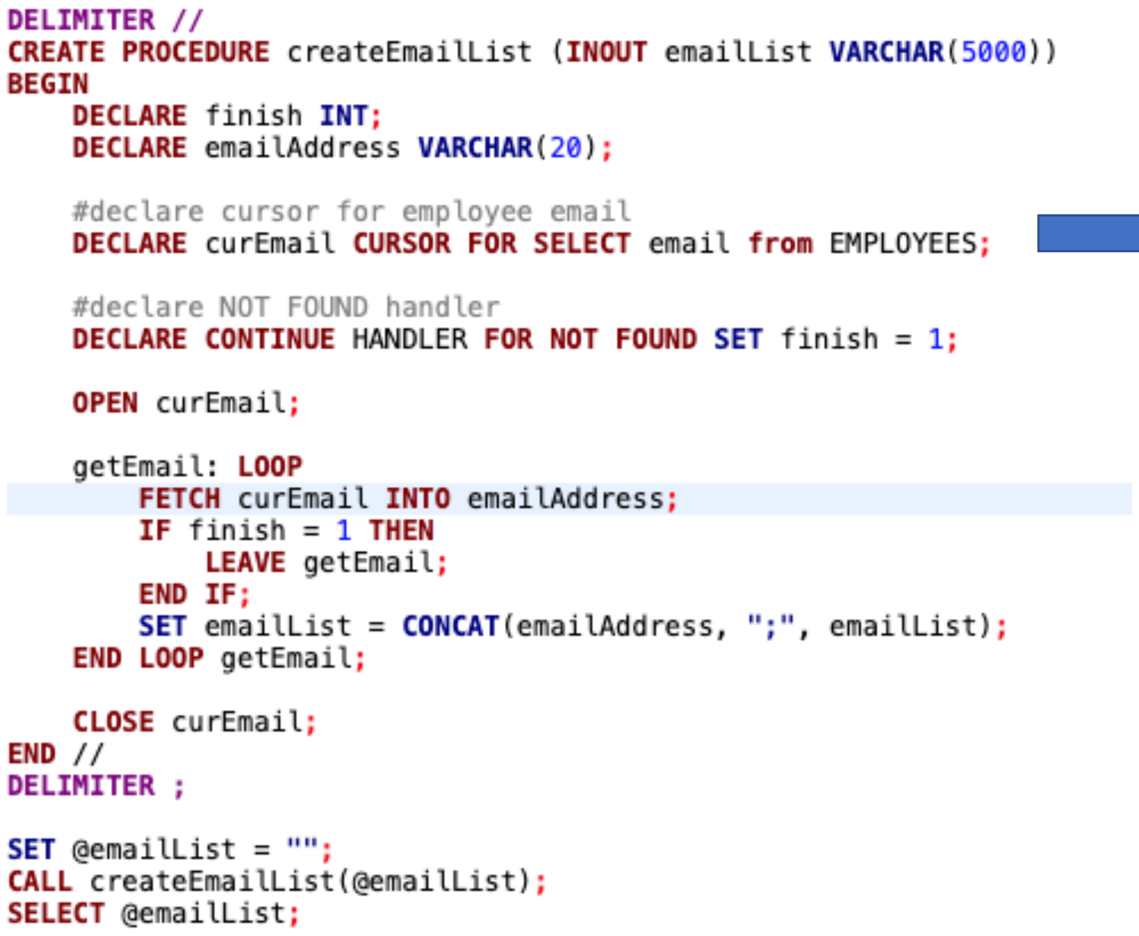


| | empNo | name | email |
|---|---|---|---|
| 1 | 1 | E1 | E1@gmail.com |
| 2 | 2 | E2 | E2@gmail.com |
| 3 | 3 | E3 | E3@gmail.com |
| 4 | 4 | E4 | E4@gmail.com |

E1@gmail.com; E2@gmail.com; E3@gmail.com; E3@gmail.com;

Source:
http://www.mysqltutorial.org/mysql-cursor/

# A procedure to create email list using cursor

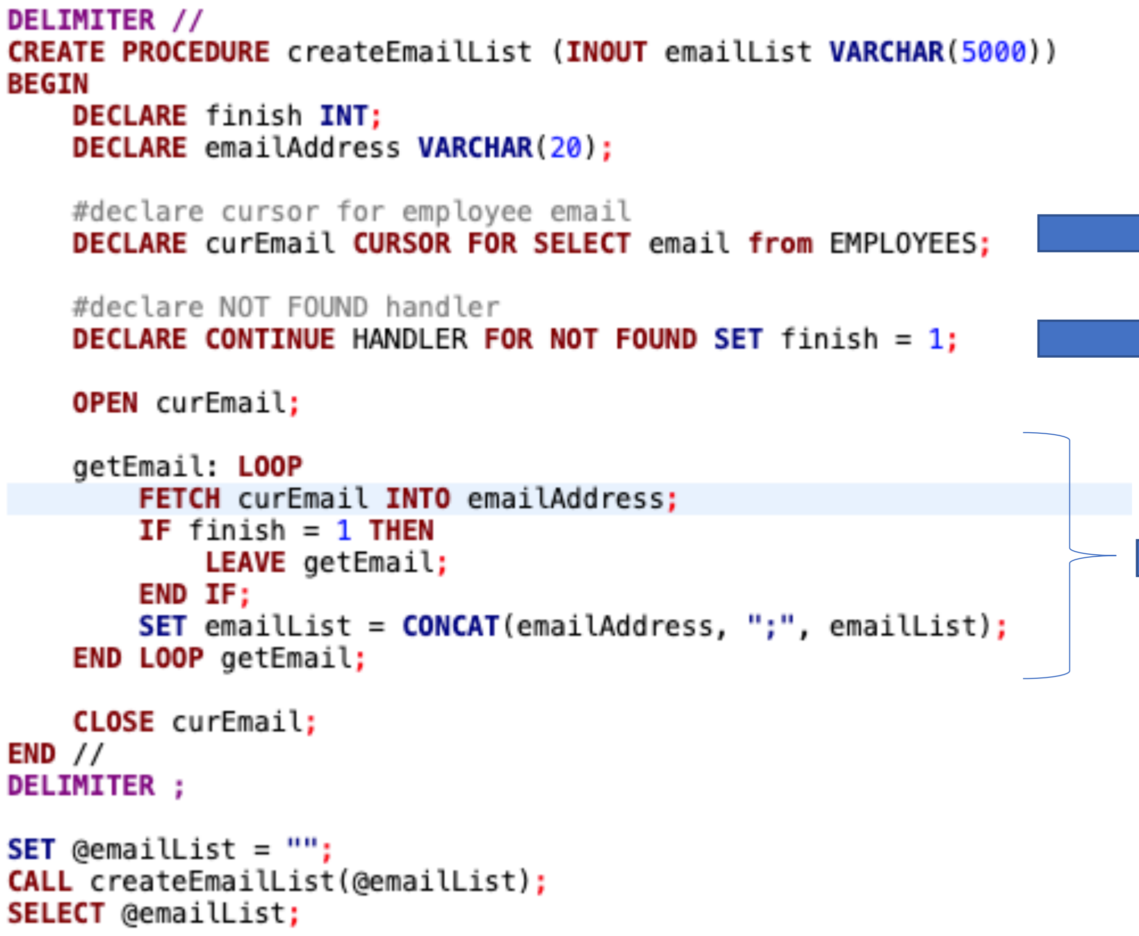


```
DELIMITER //
CREATE PROCEDURE createEmailList (INOUT emailList VARCHAR(5000))
BEGIN
    DECLARE finish INT;
    DECLARE emailAddress VARCHAR(20);

    #declare cursor for employee email
    DECLARE curEmail CURSOR FOR SELECT email from EMPLOYEES;

    #declare NOT FOUND handler
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET finish = 1;

    OPEN curEmail;

    getEmail: LOOP
        FETCH curEmail INTO emailAddress;
        IF finish = 1 THEN
            LEAVE getEmail;
        END IF;
        SET emailList = CONCAT(emailAddress, ";", emailList);
    END LOOP getEmail;

    CLOSE curEmail;
END //
DELIMITER ;

SET @emailList = "";
CALL createEmailList(@emailList);
SELECT @emailList;
```

The cursor declaration must be after any variable declaration

Source: http://www.mysqltutorial.org/mysql-cursor/

# A procedure to create email list using cursor



```
DELIMITER //
CREATE PROCEDURE createEmailList (INOUT emailList VARCHAR(5000))
BEGIN
    DECLARE finish INT;
    DECLARE emailAddress VARCHAR(20);

    #declare cursor for employee email
    DECLARE curEmail CURSOR FOR SELECT email from EMPLOYEES;

    #declare NOT FOUND handler
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET finish = 1;

    OPEN curEmail;

    getEmail: LOOP
        FETCH curEmail INTO emailAddress;
        IF finish = 1 THEN
            LEAVE getEmail;
        END IF;
        SET emailList = CONCAT(emailAddress, ";", emailList);
    END LOOP getEmail;

    CLOSE curEmail;
END //
DELIMITER ;

SET @emailList = "";
CALL createEmailList(@emailList);
SELECT @emailList;
```

The cursor declaration must be after any variable declaration
Cursor for employee email

NOT FOUND handler

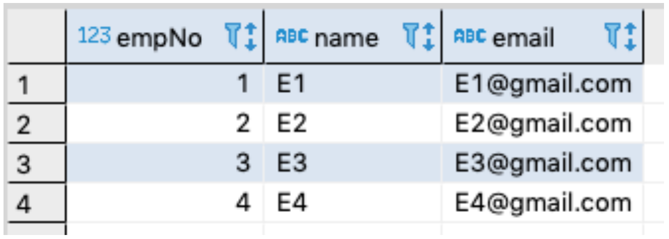
Iterate the email list, and concatenate all emails where each email is separated by a semicolon(;)

Source:
http://www.mysqltutorial.org/mysql-cursor/

# A procedure to create email list using cursor

```sql
DELIMITER //
CREATE PROCEDURE createEmailList (INOUT emailList VARCHAR(5000))
BEGIN
    DECLARE finish INT;
    DECLARE emailAddress VARCHAR(20);

    #declare cursor for employee email
    DECLARE curEmail CURSOR FOR SELECT email from EMPLOYEES;

    #declare NOT FOUND handler
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET finish = 1;

    OPEN curEmail;

    getEmail: LOOP
        FETCH curEmail INTO emailAddress;
        IF finish = 1 THEN
            LEAVE getEmail;
        END IF;
        SET emailList = CONCAT(emailAddress, ";", emailList);
    END LOOP getEmail;

    CLOSE curEmail;
END //
DELIMITER ;

SET @emailList = "";
CALL createEmailList(@emailList);
SELECT @emailList;
```

curEmail

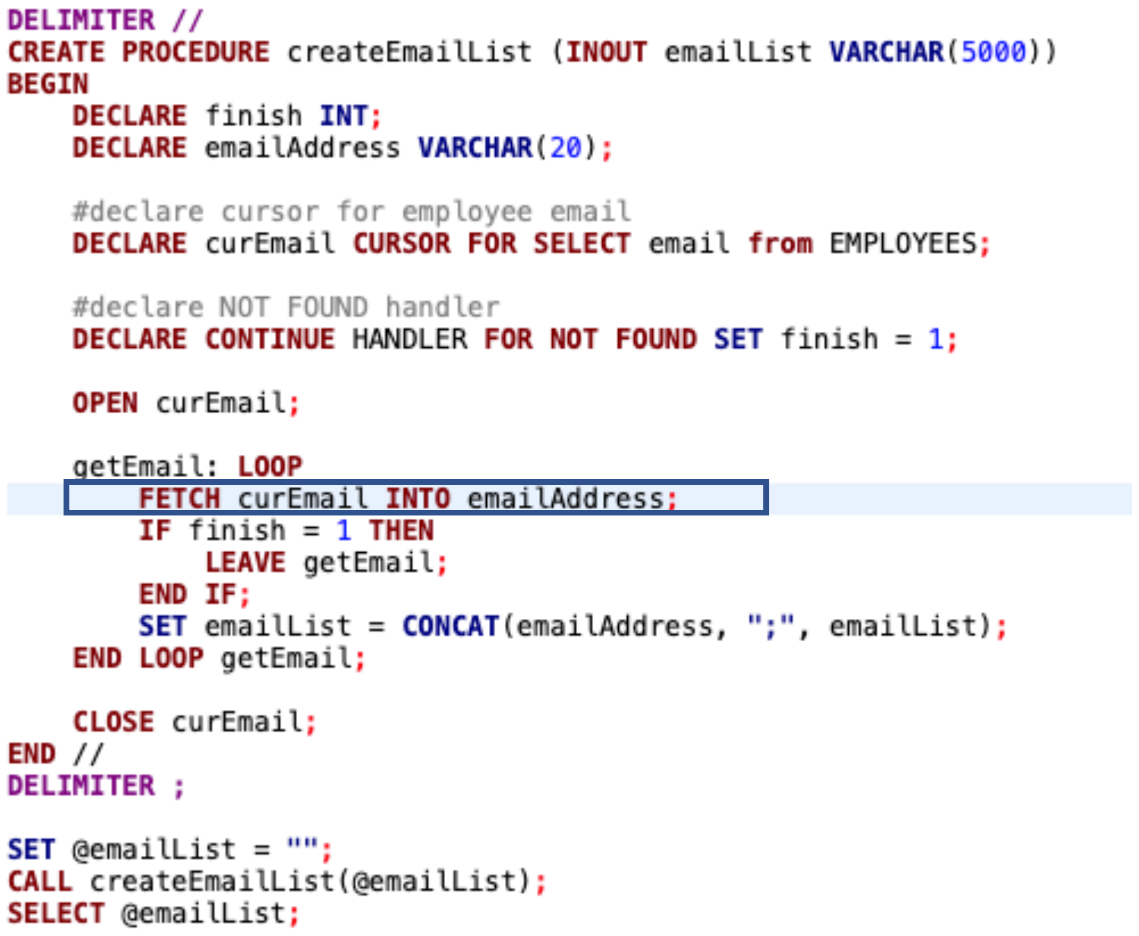


| | 123 empNo | ABC name | ABC email |
|---|---|---|---|
| 1 | 1 | E1 | E1@gmail.com |
| 2 | 2 | E2 | E2@gmail.com |
| 3 | 3 | E3 | E3@gmail.com |
| 4 | 4 | E4 | E4@gmail.com |

emailList = “”

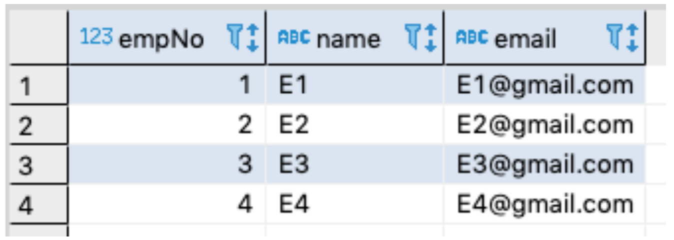
finish = 0

# A procedure to create email list using cursor



```
DELIMITER //
CREATE PROCEDURE createEmailList (INOUT emailList VARCHAR(5000))
BEGIN
    DECLARE finish INT;
    DECLARE emailAddress VARCHAR(20);

    #declare cursor for employee email
    DECLARE curEmail CURSOR FOR SELECT email from EMPLOYEES;

    #declare NOT FOUND handler
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET finish = 1;

    OPEN curEmail;

    getEmail: LOOP
        FETCH curEmail INTO emailAddress;
        IF finish = 1 THEN
            LEAVE getEmail;
        END IF;
        SET emailList = CONCAT(emailAddress, ";", emailList);
    END LOOP getEmail;

    CLOSE curEmail;
END //
DELIMITER ;

SET @emailList = "";
CALL createEmailList(@emailList);
SELECT @emailList;
```

curEmail



| | empNo | name | email |
|---|---|---|---|
| 1 | 1 | E1 | E1@gmail.com |
| 2 | 2 | E2 | E2@gmail.com |
| 3 | 3 | E3 | E3@gmail.com |
| 4 | 4 | E4 | E4@gmail.com |

emailList = “”

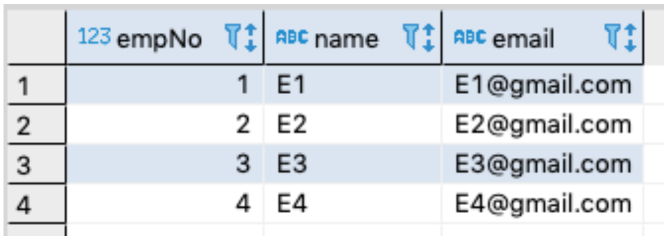
finish = 0

emailAddress = E1@gmail.com

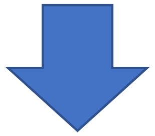
# A procedure to create email list using cursor

```
DELIMITER //
CREATE PROCEDURE createEmailList (INOUT emailList VARCHAR(5000))
BEGIN
    DECLARE finish INT;
    DECLARE emailAddress VARCHAR(20);

    #declare cursor for employee email
    DECLARE curEmail CURSOR FOR SELECT email from EMPLOYEES;

    #declare NOT FOUND handler
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET finish = 1;

    OPEN curEmail;

    getEmail: LOOP
        FETCH curEmail INTO emailAddress;
        IF finish = 1 THEN
            LEAVE getEmail;
        END IF;
        SET emailList = CONCAT(emailAddress, ";", emailList);
    END LOOP getEmail;

    CLOSE curEmail;
END //
DELIMITER ;

SET @emailList = "";
CALL createEmailList(@emailList);
SELECT @emailList;
```

curEmail



| | empNo | name | email |
|---|---|---|---|
| 1 | 1 | E1 | E1@gmail.com |
| 2 | 2 | E2 | E2@gmail.com |
| 3 | 3 | E3 | E3@gmail.com |
| 4 | 4 | E4 | E4@gmail.com |

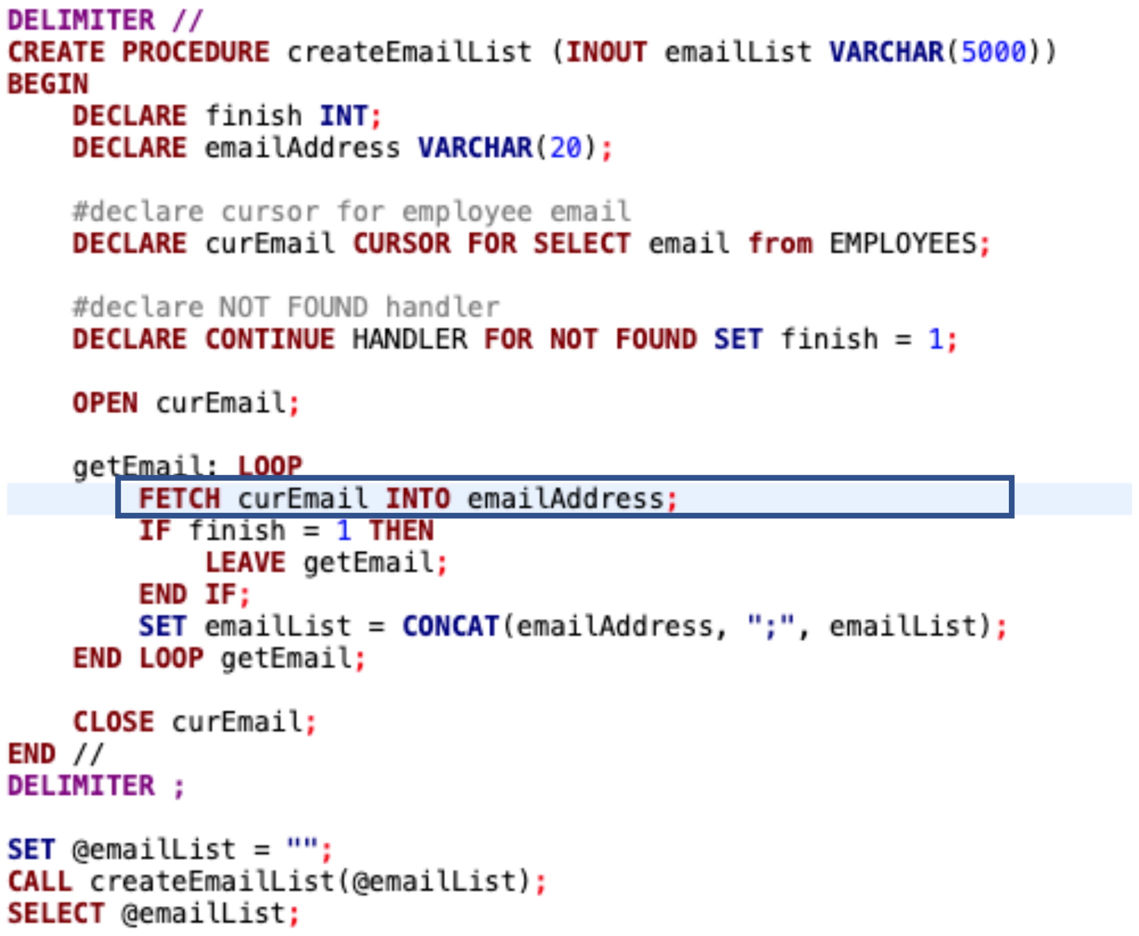


finish = 0

emailAddress = E1@gmail.com

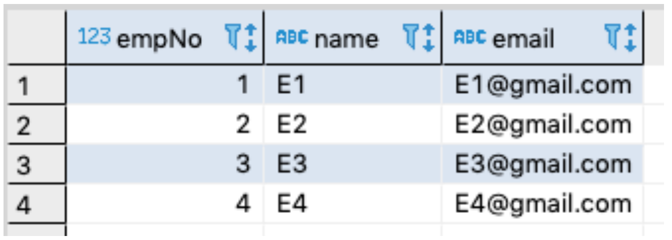
emailList = “E1@gmail.com;”

# A procedure to create email list using cursor



```
DELIMITER //
CREATE PROCEDURE createEmailList (INOUT emailList VARCHAR(5000))
BEGIN
    DECLARE finish INT;
    DECLARE emailAddress VARCHAR(20);

    #declare cursor for employee email
    DECLARE curEmail CURSOR FOR SELECT email from EMPLOYEES;

    #declare NOT FOUND handler
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET finish = 1;

    OPEN curEmail;

    getEmail: LOOP
        FETCH curEmail INTO emailAddress;
        IF finish = 1 THEN
            LEAVE getEmail;
        END IF;
        SET emailList = CONCAT(emailAddress, ";", emailList);
    END LOOP getEmail;

    CLOSE curEmail;
END //
DELIMITER ;

SET @emailList = "";
CALL createEmailList(@emailList);
SELECT @emailList;
```

curEmail



| | 123 empNo | ABC name | ABC email |
|---|---|---|---|
| 1 | 1 | E1 | E1@gmail.com |
| 2 | 2 | E2 | E2@gmail.com |
| 3 | 3 | E3 | E3@gmail.com |
| 4 | 4 | E4 | E4@gmail.com |

finish = 0

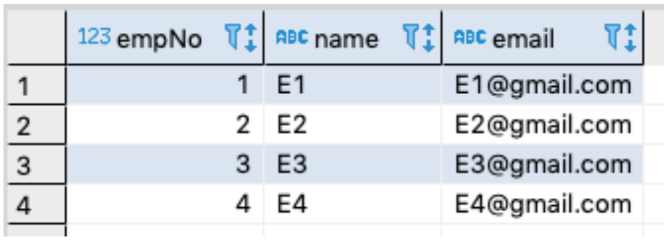
emailAddress = E2@gmail.com

emailList = “E1@gmail.com;”

# A procedure to create email list using cursor

```sql
DELIMITER //
CREATE PROCEDURE createEmailList (INOUT emailList VARCHAR(5000))
BEGIN
    DECLARE finish INT;
    DECLARE emailAddress VARCHAR(20);

    #declare cursor for employee email
    DECLARE curEmail CURSOR FOR SELECT email from EMPLOYEES;

    #declare NOT FOUND handler
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET finish = 1;

    OPEN curEmail;

    getEmail: LOOP
        FETCH curEmail INTO emailAddress;
        IF finish = 1 THEN
            LEAVE getEmail;
        END IF;
        SET emailList = CONCAT(emailAddress, ";", emailList);
    END LOOP getEmail;

    CLOSE curEmail;
END //
DELIMITER ;

SET @emailList = "";
CALL createEmailList(@emailList);
SELECT @emailList;
```

curEmail



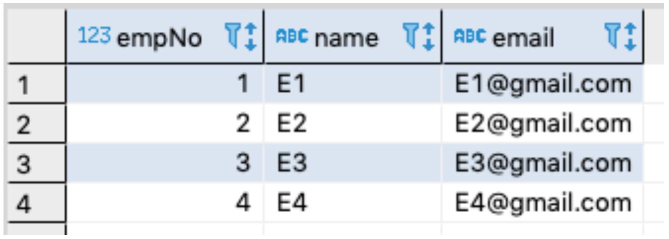
| | empNo | name | email |
|---|---|---|---|
| 1 | 1 | E1 | E1@gmail.com |
| 2 | 2 | E2 | E2@gmail.com |
| 3 | 3 | E3 | E3@gmail.com |
| 4 | 4 | E4 | E4@gmail.com |

finish = 0
emailAddress = E2@gmail.com
emailList = “E1@gmail.com; E2@gmail.com”

# A procedure to create email list using cursor

```
DELIMITER //
CREATE PROCEDURE createEmailList (INOUT emailList VARCHAR(5000))
BEGIN
    DECLARE finish INT;
    DECLARE emailAddress VARCHAR(20);

    #declare cursor for employee email
    DECLARE curEmail CURSOR FOR SELECT email from EMPLOYEES;

    #declare NOT FOUND handler
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET finish = 1;

    OPEN curEmail;

    getEmail: LOOP
        FETCH curEmail INTO emailAddress;
        IF finish = 1 THEN
            LEAVE getEmail;
        END IF;
        SET emailList = CONCAT(emailAddress, ";", emailList);
    END LOOP getEmail;

    CLOSE curEmail;
END //
DELIMITER ;

SET @emailList = "";
CALL createEmailList(@emailList);
SELECT @emailList;
```

curEmail



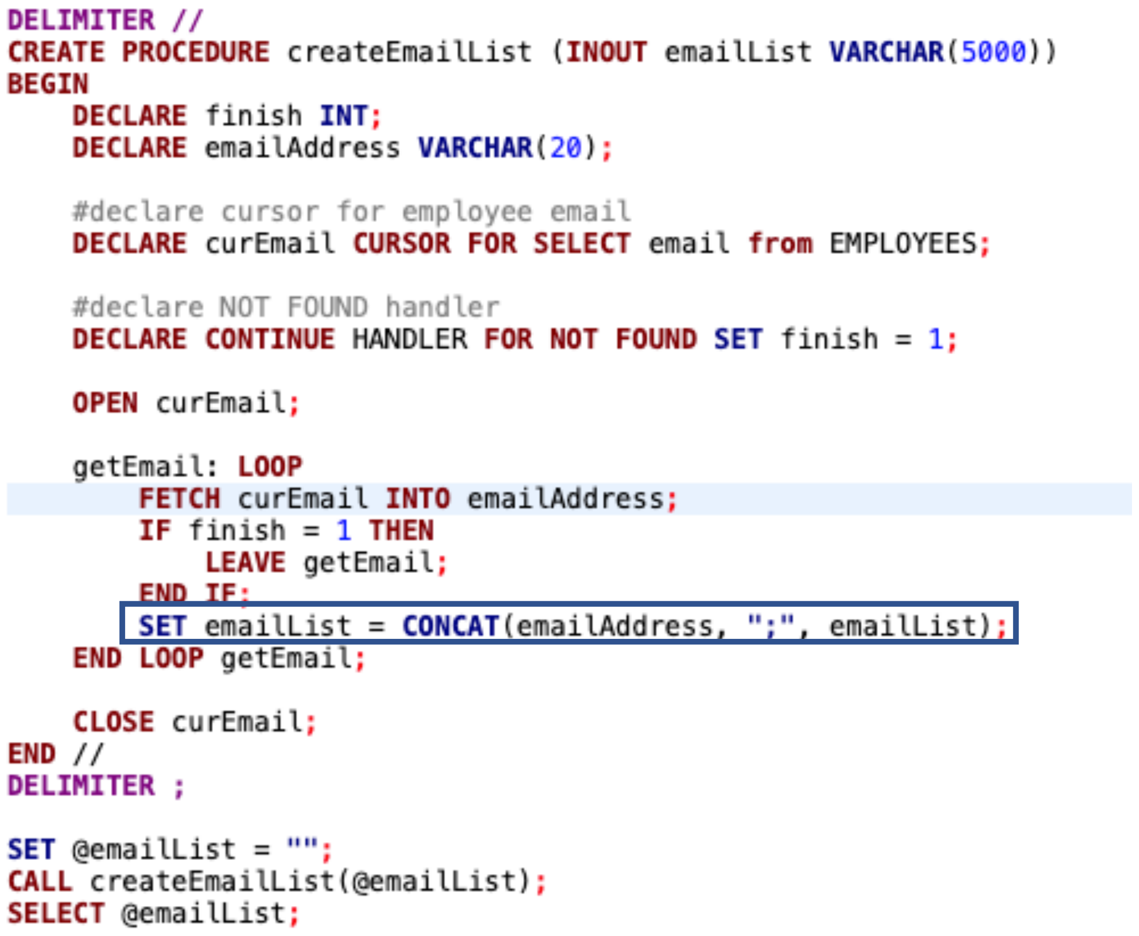
| | empNo | name | email |
|---|---|---|---|
| 1 | 1 | E1 | E1@gmail.com |
| 2 | 2 | E2 | E2@gmail.com |
| 3 | 3 | E3 | E3@gmail.com |
| 4 | 4 | E4 | E4@gmail.com |

finish = 0

emailAddress = E3@gmail.com

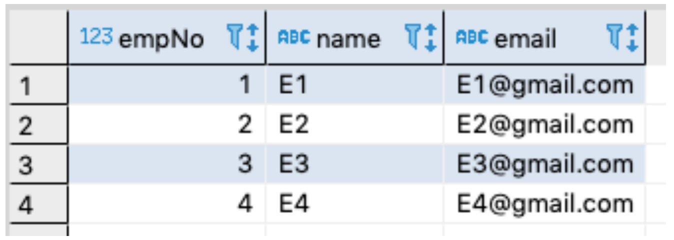
emailList = “E1@gmail.com; E2@gmail.com”

# A procedure to create email list using cursor



```
DELIMITER //
CREATE PROCEDURE createEmailList (INOUT emailList VARCHAR(5000))
BEGIN
    DECLARE finish INT;
    DECLARE emailAddress VARCHAR(20);

    #declare cursor for employee email
    DECLARE curEmail CURSOR FOR SELECT email from EMPLOYEES;

    #declare NOT FOUND handler
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET finish = 1;

    OPEN curEmail;

    getEmail: LOOP
        FETCH curEmail INTO emailAddress;
        IF finish = 1 THEN
            LEAVE getEmail;
        END IF;
        SET emailList = CONCAT(emailAddress, ";", emailList);
    END LOOP getEmail;

    CLOSE curEmail;
END //
DELIMITER ;

SET @emailList = "";
CALL createEmailList(@emailList);
SELECT @emailList;
```

curEmail



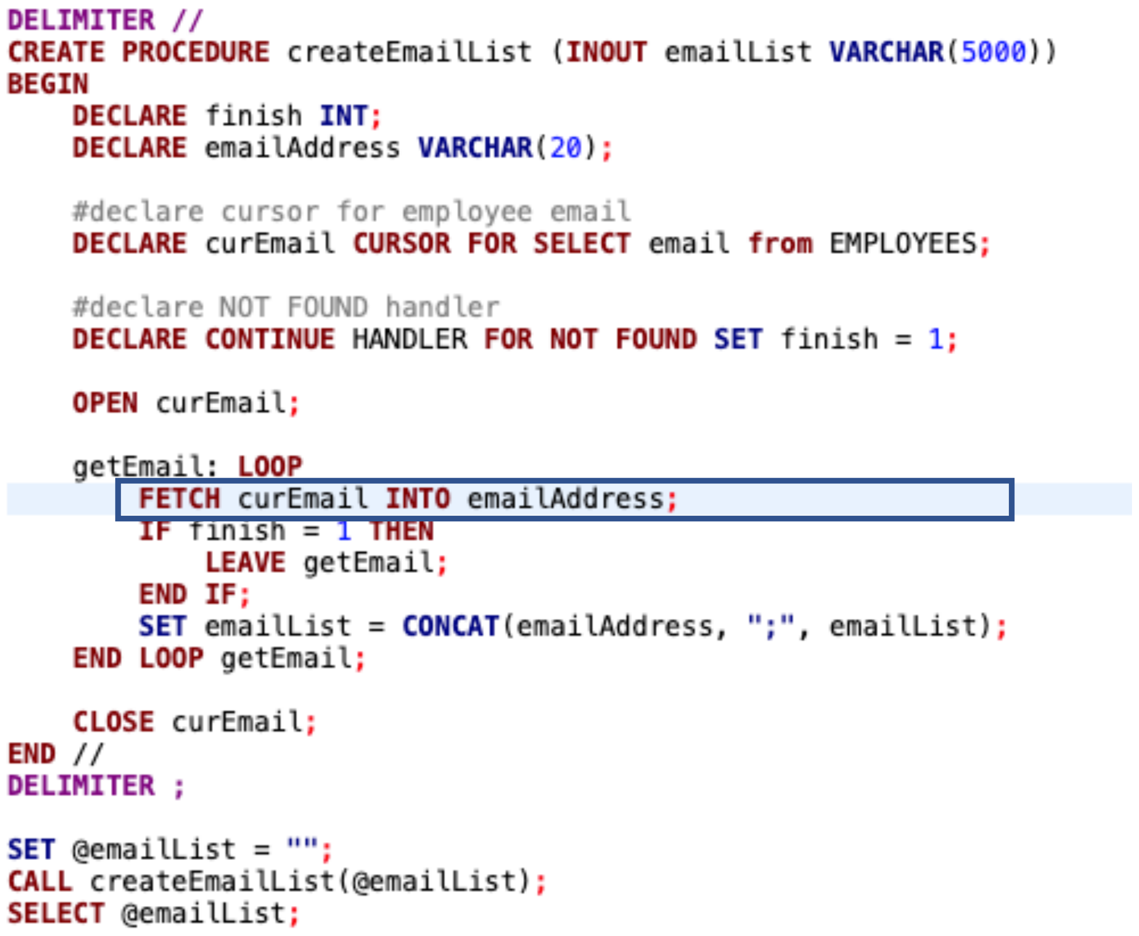
| | empNo | name | email |
|---|---|---|---|
| 1 | 1 | E1 | E1@gmail.com |
| 2 | 2 | E2 | E2@gmail.com |
| 3 | 3 | E3 | E3@gmail.com |
| 4 | 4 | E4 | E4@gmail.com |

finish = 0

emailAddress = E3@gmail.com

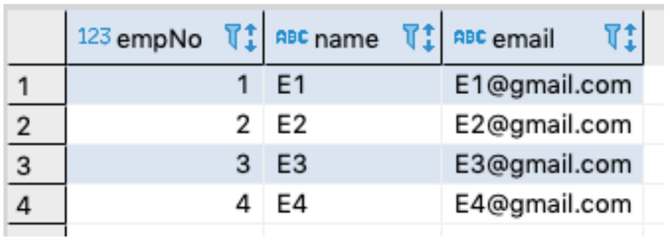
emailList = "E1@gmail.com; E2@gmail.com;
E3@gmail.com;
"

# A procedure to create email list using cursor

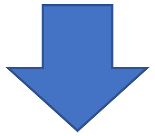


```
DELIMITER //
CREATE PROCEDURE createEmailList (INOUT emailList VARCHAR(5000))
BEGIN
    DECLARE finish INT;
    DECLARE emailAddress VARCHAR(20);

    #declare cursor for employee email
    DECLARE curEmail CURSOR FOR SELECT email from EMPLOYEES;

    #declare NOT FOUND handler
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET finish = 1;

    OPEN curEmail;

    getEmail: LOOP
        FETCH curEmail INTO emailAddress;
        IF finish = 1 THEN
            LEAVE getEmail;
        END IF;
        SET emailList = CONCAT(emailAddress, ";", emailList);
    END LOOP getEmail;

    CLOSE curEmail;
END //
DELIMITER ;

SET @emailList = "";
CALL createEmailList(@emailList);
SELECT @emailList;
```

curEmail



| | 123 empNo | ABC name | ABC email |
|---|---|---|---|
| 1 | 1 | E1 | E1@gmail.com |
| 2 | 2 | E2 | E2@gmail.com |
| 3 | 3 | E3 | E3@gmail.com |
| 4 | 4 | E4 | E4@gmail.com |



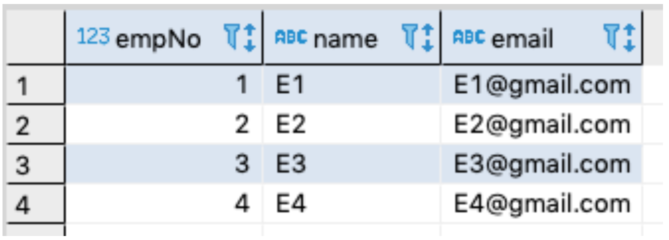
finish = 0

emailAddress = E4@gmail.com

emailList = “E1@gmail.com; E2@gmail.com;
E3@gmail.com;
”

# A procedure to create email list using cursor

```
DELIMITER //
CREATE PROCEDURE createEmailList (INOUT emailList VARCHAR(5000))
BEGIN
    DECLARE finish INT;
    DECLARE emailAddress VARCHAR(20);

    #declare cursor for employee email
    DECLARE curEmail CURSOR FOR SELECT email from EMPLOYEES;

    #declare NOT FOUND handler
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET finish = 1;

    OPEN curEmail;

    getEmail: LOOP
        FETCH curEmail INTO emailAddress;
        IF finish = 1 THEN
            LEAVE getEmail;
        END IF;
        SET emailList = CONCAT(emailAddress, ";", emailList);
    END LOOP getEmail;

    CLOSE curEmail;
END //
DELIMITER ;

SET @emailList = "";
CALL createEmailList(@emailList);
SELECT @emailList;
```



curEmail

| | empNo | name | email |
|---|---|---|---|
| 1 | 1 | E1 | E1@gmail.com |
| 2 | 2 | E2 | E2@gmail.com |
| 3 | 3 | E3 | E3@gmail.com |
| 4 | 4 | E4 | E4@gmail.com |

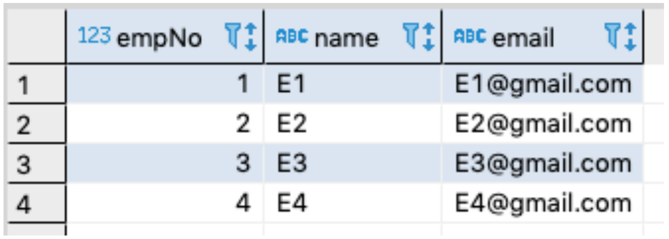
finish = 0

emailAddress = E4@gmail.com

emailList = “E1@gmail.com; E2@gmail.com; E3@gmail.com; E4@gmail.com ”

# A procedure to create email list using cursor

```sql
DELIMITER //
CREATE PROCEDURE createEmailList (INOUT emailList VARCHAR(5000))
BEGIN
    DECLARE finish INT;
    DECLARE emailAddress VARCHAR(20);

    #declare cursor for employee email
    DECLARE curEmail CURSOR FOR SELECT email from EMPLOYEES;

    #declare NOT FOUND handler
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET finish = 1;

    OPEN curEmail;

    getEmail: LOOP
        FETCH curEmail INTO emailAddress;
        IF finish = 1 THEN
            LEAVE getEmail;
        END IF;
        SET emailList = CONCAT(emailAddress, ";", emailList);
    END LOOP getEmail;

    CLOSE curEmail;
END //
DELIMITER ;

SET @emailList = "";
CALL createEmailList(@emailList);
SELECT @emailList;
```



| | empNo | name | email |
|---|---|---|---|
| 1 | 1 | E1 | E1@gmail.com |
| 2 | 2 | E2 | E2@gmail.com |
| 3 | 3 | E3 | E3@gmail.com |
| 4 | 4 | E4 | E4@gmail.com |

curEmail

finish = 1

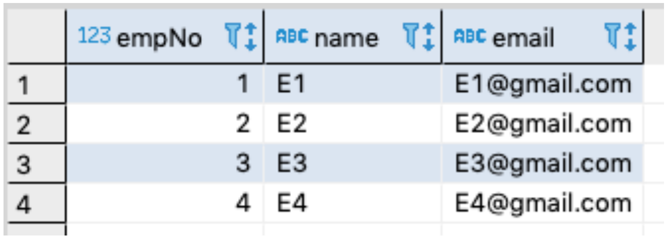
emailAddress = E4@gmail.com

emailList = "E1@gmail.com; E2@gmail.com;
E3@gmail.com; E4@gmail.com
"

# A procedure to create email list using cursor

```
DELIMITER //
CREATE PROCEDURE createEmailList (INOUT emailList VARCHAR(5000))
BEGIN
    DECLARE finish INT;
    DECLARE emailAddress VARCHAR(20);

    #declare cursor for employee email
    DECLARE curEmail CURSOR FOR SELECT email from EMPLOYEES;

    #declare NOT FOUND handler
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET finish = 1;

    OPEN curEmail;

    getEmail: LOOP
        FETCH curEmail INTO emailAddress;
        IF finish = 1 THEN
            LEAVE getEmail;
        END IF;
        SET emailList = CONCAT(emailAddress, ";", emailList);
    END LOOP getEmail;

    CLOSE curEmail;
END //
DELIMITER ;

SET @emailList = "";
CALL createEmailList(@emailList);
SELECT @emailList;
```

**Finish this loop**



| | empNo | name | email |
|---|---|---|---|
| 1 | 1 | E1 | E1@gmail.com |
| 2 | 2 | E2 | E2@gmail.com |
| 3 | 3 | E3 | E3@gmail.com |
| 4 | 4 | E4 | E4@gmail.com |

curEmail

finish = 1

emailAddress = E4@gmail.com

emailList = "E1@gmail.com; E2@gmail.com; E3@gmail.com; E4@gmail.com"

## Integrity Constraints

- An Integrity Constraint (IC) describes conditions that every legal instance of a relation must satisfy
- To disallow inserts/deletes/updates that violate IC’s
- Types of IC’s:  Domain constraints, primary key constraints, foreign key constraints, non-null, general constraints

## An Example via CHECK Clause

```
CREATE TABLE  Students (
        sid  INT,
        sname  VARCHAR(10),
        rating  INT,
        age  INT,
        PRIMARY KEY  (sid),
        CONSTRAINT checkRating
        CHECK  (rating >= 1 AND rating <= 10 )
)
```

```
  INSERT INTO Students VALUES(1, 'Jones', 9, 19);
  INSERT INTO Students VALUES(2, 'Smith', 7, 19);
X INSERT INTO Students VALUES(2, 'Peter', 19, 19);
```

## ASSERTION Example
## Constraints Over Multiple Relations

- Consider a very small school: the count of students and professors should be less than 500
- The following is a poor integrity test as it is associated with one relation (the Students table could be empty and thus the integrity rule is never checked!
- Disassociate from the Students table

```
CREATE TABLE  Students (
            sid  INTEGER,
            sname  VARCHAR(10),
            rating  INT,
            age  INT,
            PRIMARY KEY  (sid),
            CHECK  (
            (SELECT COUNT (S.sid) FROM Students S) +
            (SELECT COUNT (P.pid) FROM Profesor P) < 500 )
)
```

## ASSERTION Example
## Constraints Over Multiple Relations

```
CREATE ASSERTION smallSchool
CHECK  (
        (SELECT COUNT (S.sid) FROM Stu S) +
        (SELECT COUNT (P.pid) FROM Prof P) < 500
)
```

## ASSERTION Example
## The KEY Constraint

```
CREATE ASSERTION Key
CHECK  (
        (SELECT COUNT (DISTINCT sid) FROM Stu) =
        (SELECT COUNT (*) FROM Stu)
);
```

- Note: ASSERTION is in standard SQL but not implemented
- Unfortunately, MySQL does not support ASSERTIONS, but we can uses triggers  to implement this functionality

## Triggers

- A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs
- A trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated

When **event** occurs, check **condition**; if true do **action**

## Advantages

- To move application logic and business rules into database
- Allows more functionality for DBAs to establish vital constraints/rules of applications
- Rules managed in some central “place”
- Rules automatically enforced by DBMS, no matter which applications later come on line

## The Event-Condition-Action Model

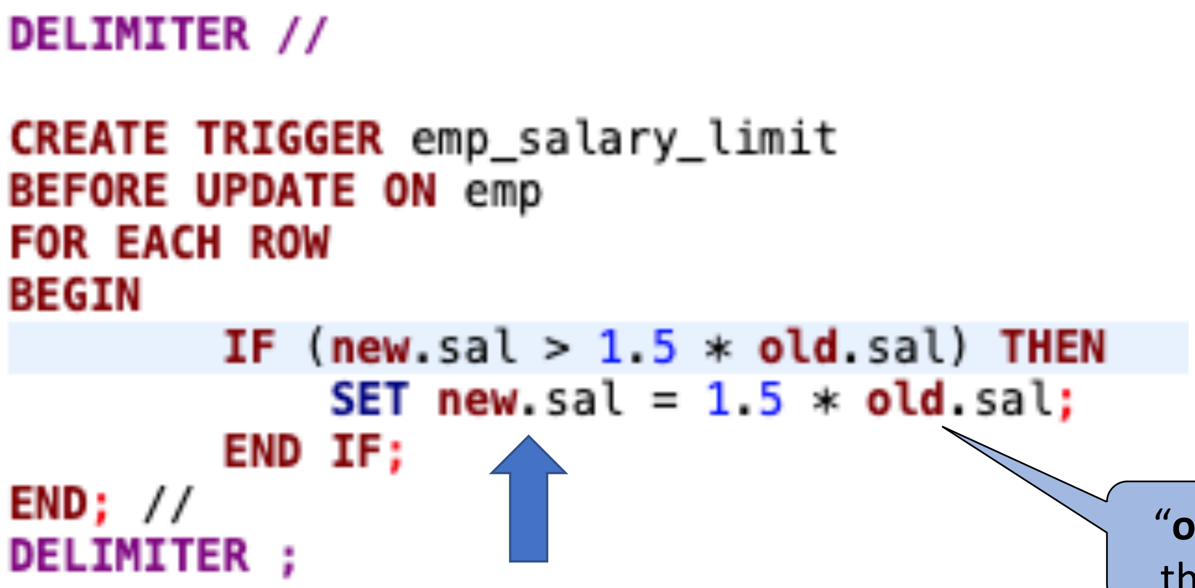
- Actions may apply before or after the triggering event is executed
- An SQL statement may change several rows
  - Apply action once per SQL statement
  - Apply action for each row changed by SQL statement

## The Company Database

```
EMPLOYEE(Name, SSN, Salary, DNO, SupervisorSSN, JobCode)
DEPARTMENT(DNO, TotalSalary, ManagerSSN)
STARTING_PAY(JobCode, StartPay)
```

1. Limit all salary increases to 50%.
2. Enforce policy that salaries may never decrease.
3. Maintain `TotalSalary` in `DEPARTMENT` relation as employees and their salaries change.
4. Inform a supervisor whenever a supervisee's salary becomes larger than the supervisor's.
5. All new hires for a given job code get the same starting salary, which is available in the `STARTING_PAY` table.

## Limit all salary increases to 50%



```
DELIMITER //

CREATE TRIGGER emp_salary_limit
BEFORE UPDATE ON emp
FOR EACH ROW
BEGIN
        IF (new.sal > 1.5 * old.sal) THEN
            SET new.sal = 1.5 * old.sal;
        END IF;
END; //
DELIMITER ;
```

“**new**” refers to the new tuple.

“**old**” refers to the old tuple.