



# Database Updates and Stored Procedures

**Abdu Alawini**

University of Illinois at Urbana-Champaign

CS411: Database Systems

# Learning Objectives

After this lecture, you should be able to:

- Write commands to update database instance and schema
- Define and execute Stored Procedures in MySQL



# Database Modifications

- A modification command does not return a result the way a query does, but it changes the database in some way.
- There are two kinds of database modifications:
  1. *Instance Modifications*
    - *Insert* a tuple or tuples.
    - *Delete* a tuple or tuples.
    - *Update* the value(s) of an existing tuple or tuples.
  2. Schema Modifications
    - Add/Drop/Modify column
    - Drop Table/View



# Insertion

- To insert a single tuple:

```
INSERT INTO <relation>
```

```
VALUES ( <list of values> );
```

- Example: add to Likes (customer, drink) the fact that Sally likes Latte.

```
INSERT INTO Likes
```

```
VALUES ('Sally', 'Latte');
```



# Specifying Attributes in Insert

```
INSERT INTO Likes  
VALUES ('Sally', 'Latte');
```

- BUT: this assumes that we remember the order of attributes of Likes
- Instead: we can be explicit: to insert Sally into Likes (customer, drink):

```
INSERT INTO Likes (drink, customer)  
VALUES ('Latte', 'Sally');
```

- Can also add multiple tuples separated by commas



# Specifying Attributes in INSERT

Overall, two reasons to specify attributes in the INSERT statement:

1. We may have forgotten the standard order of attributes for the relation.
2. We don't have values for all attributes, and we want the system to fill in missing components with NULL or a default value.

*simply omit the ones you don't want to insert*

# Inserting Many Tuples

We may insert the entire result of a query into a relation, using the form:

```
INSERT INTO <relation>
( <subquery> );
```

E.g., `INSERT INTO Drinks(name)
SELECT drink FROM Sells;`



# Deletion

- To delete tuples satisfying a condition from some relation:

```
DELETE FROM <relation>
```

```
WHERE <condition>;
```



## Example: Deletion

- Delete from Likes (customer, drink) the fact that Sally likes Latte:

```
DELETE FROM Likes  
WHERE customer = 'Sally' AND  
drink = 'Latte';
```



## Example: Delete all Tuples

- Make the relation Likes empty:

```
DELETE FROM Likes;
```

- Note no WHERE clause needed.
- Table is not deleted: use the DROP TABLE statement instead



## Example: Delete Many Tuples

- Delete from Drinks(name, manf) all drinks manufactured by Starbucks

```
DELETE FROM Drinks  
WHERE name = 'Starbucks';
```



## Another Example: Delete Many Tuples

- Delete from Drinks (name, manf) all drinks for which there is another drink by the same manufacturer.

```
DELETE FROM Drinks  
WHERE name IN (  
  
    SELECT b1.name  
        FROM Drinks b1, Drinks b2  
       WHERE b1.manf = b2.manf AND  
             b1.name <> b2.name);
```

Drinks with the same  
manufacturer and  
different names





# Updates

- To change certain attributes in certain tuples of a relation:

UPDATE <relation>

SET <list of attribute assignments>

WHERE <condition on tuples>;

# Example: Update

- Change Fred's phone number to 555-1212:

```
UPDATE Customer  
SET phone = '555-1212'  
WHERE name = 'Fred';
```

- Add area code '217' to Fred's phone number:

```
UPDATE Customer  
SET phone = '(217)' || phone  
WHERE name = 'Fred';
```



## What about Multiple Users?

- What happens if multiple users update + delete at the same time?
- This requires management of concurrent operations
- We'll talk about **concurrency control** later in the semester

# Outline

- ✓ Database Updates
- Stored Procedures



# Stored Procedures in MySQL

- A Stored Procedure is a set of SQL statements (with an assigned name) stored in the DBMS and can be called by multiple programs.
- Stored Procedure Syntax:

```
CREATE PROCEDURE sp_name
    (proc_parameter,...)
BEGIN
    -- execution code
END;
```

*proc\_parameter:*

[ IN | OUT | INOUT ] *param\_name param\_type*

- IN parameters for passing values into the procedure,
- OUT parameters for passing value back from procedure to the calling program
- INOUT: is a combination of IN and OUT parameters.



## Example

Write a Stored Procedure that returns the FirstName, LastName, and Average GPA for each student

```
DELIMITER //
CREATE PROCEDURE GetAverageScore ()
BEGIN
    SELECT s.FirstName, s.LastName, AVG(Score) AS avgScore
    FROM Enrollments e JOIN Students s
        ON (e.NetId = s.NetId)
    GROUP BY s.NetId;
END //
DELIMITER ;
```

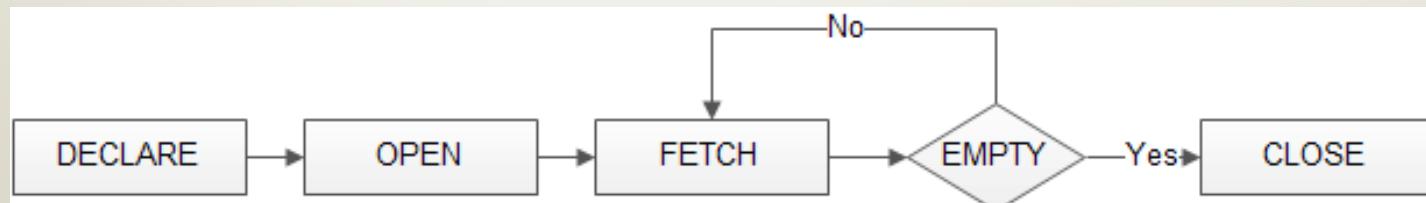
Calling the procedure:

```
CALL GetAverageScore();
```



# Stored Procedures in MySQL

- In stored procedures, you can:
  - declare variables
  - use conditional IF-THEN-ELSE or loops such as WHILE and REPEAT statements
  - use cursors: A cursor is used to iterate through a set of rows returned by a query so that we can process each individual row.
- how does MySQL cursor work?



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



# Using Variables in MySQL SP

Define a stored procedure that takes a department name and returns the total number of students in that department.

```
DELIMITER //
CREATE PROCEDURE GetTotalStds (IN dept VARCHAR(30))
BEGIN
    DECLARE totalStds INT DEFAULT 0
    SELECT COUNT(*)
    INTO totalStds
    FROM Students
    WHERE Department= dept;

    SELECT totalStds;
END //
DELIMITER ;
```

Calling the procedure: CALL GetTotalStds('CS');



# Passing values IN and OUT of SPs

Define a stored procedure that takes a department and returns the total number of students in that department.

```
DELIMITER //
CREATE PROCEDURE GetTotalStds (
    IN dept VARCHAR(30),
    OUT total INT)
BEGIN
    SELECT COUNT(*)
    INTO total
    FROM Students
    WHERE Department= dept;
END //
DELIMITER ;
```

Calling the procedure: CALL GetTotalStds('CS',@total);  
Getting the result: SELECT @total;



## SP Example using Cursor

- Suppose we want to compute the average GPA for students per department and save the result in a new table DeptAvgGPA(deptName, AverageScore)
- **STEP 1: Change the delimiter from ; to //**

```
[mysql> DELIMITER //
```



# SP Example using Cursor

- Suppose we want to compute the average students' GPA per department and save the result in a new table DeptAvgGPA(deptName, AverageScore)

## STEP 2: Define the stored procedure

```
mysql> CREATE PROCEDURE deptAvgGPA()
->     BEGIN
->         DECLARE done int default 0;
->         DECLARE currdept VARCHAR(30);
->         DECLARE deptcur CURSOR FOR SELECT DISTINCT department FROM students;
->         DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
->
->         DROP TABLE IF EXISTS deptAvgGPA;
->
->         CREATE TABLE deptAvgGPA (
->             deptName VARCHAR(30),
->             avgGPA REAL
->         );
->
->         OPEN deptcur;
->
->         REPEAT
->             FETCH deptcur INTO currdept;
->             INSERT INTO deptAvgGPA
->                 (SELECT department, AVG(GPA) FROM students WHERE department = currdept);
->             UNTIL done
->             END REPEAT;
->
->             close deptcur;
->         END //
Query OK, 0 rows affected (0.01 sec)
```



# SP Example using Cursor

- Suppose we want to compute the average score for students per department and save the result in a new table DeptAvgGPA(deptName, AverageScore)

**STEP 3: Changing the delimiter back to semi colon (;**

```
mysql> DELIMITER ;
```

**STEP 4: calling stored procedure**

```
[mysql> call deptAvgGPA();
Query OK, 1 row affected (0.06 sec)
```

**Viewing the results:**

```
[mysql> select * from deptAvgGPA;
+-----+-----+
| deptName | avgGPA
+-----+-----+
| ECE      | 2.6346153846153846 |
| CS       | 2.073529411764706  |
| ME       | 2.4705882352941178  |
| IS       | 2.142857142857143   |
| ECON     | 2.2916666666666665   |
| CE       | 2.4558823529411766  |
| EM       | 2.525                  |
| EM       | 2.525                  |
+-----+-----+
```



# MySQL Stored Procedures Tutorial

<http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx>