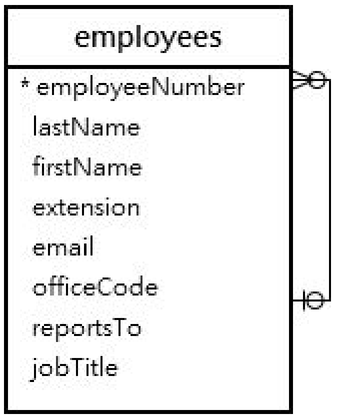
Triggers

- In MySQL, a trigger is a stored program invoked automatically in response to an event such as **INSERT**, **UPDATE**, or **DELETE** that occurs in the associated table.

- It can be invoked **before** or **after** the event. For example, you can define a trigger that is invoked automatically before a new row is inserted into a table.

Create Trigger:

Let's start creating a trigger to log the changes of employee table.



First, create a new table named employees_audit to keep the changes to the employees table:

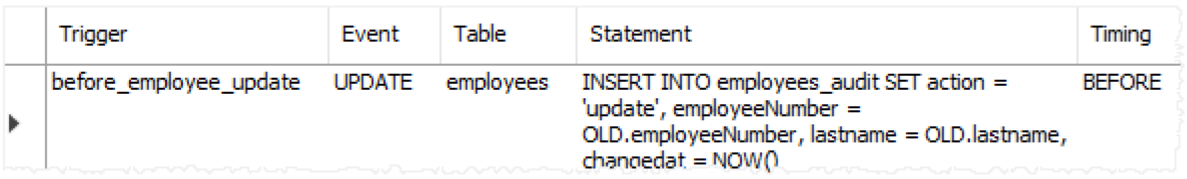
```
CREATE TABLE employees_audit (
id INT AUTO_INCREMENT PRIMARY KEY,
employeeNumber INT NOT NULL,
lastname VARCHAR(50) NOT NULL,
changedate DATETIME DEFAULT NULL,
action VARCHAR(50) DEFAULT NULL
);
```

Next, create a BEFORE UPDATE trigger that is invoked before a change is made to the employees table.

```
DELIMITER //
CREATE TRIGGER before_employee_update
BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
INSERT INTO employees_audit
SET action = 'update',
employeeNumber = OLD.employeeNumber,
lastname = OLD.lastname,
changedat = NOW();
END //
DELIMITER ;
```

- we used the OLD keyword to access values of the columns employeeNumber and lastname of the row affected by the trigger.

- Use **SHOW TRIGGERS** to show all the triggers in the current database.



Trigger	Event	Table	Statement	Timing
before_employee_update	UPDATE	employees	INSERT INTO employees_audit SET action = 'update', employeeNumber = OLD.employeeNumber, lastname = OLD.lastname, changedat = NOW()	BEFORE

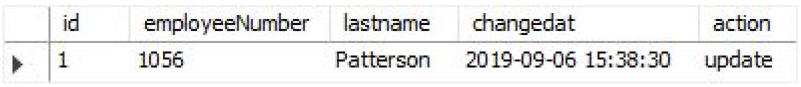
After that, update a row in the employees table:

```
UPDATE employees SET lastName = 'Phan' WHERE
employeeNumber = 1056;
```

Query the employees_audit table to check if the trigger was fired by the UPDATE statement:

```
SELECT * FROM employees_audit;
```

Following is the output for the above query.



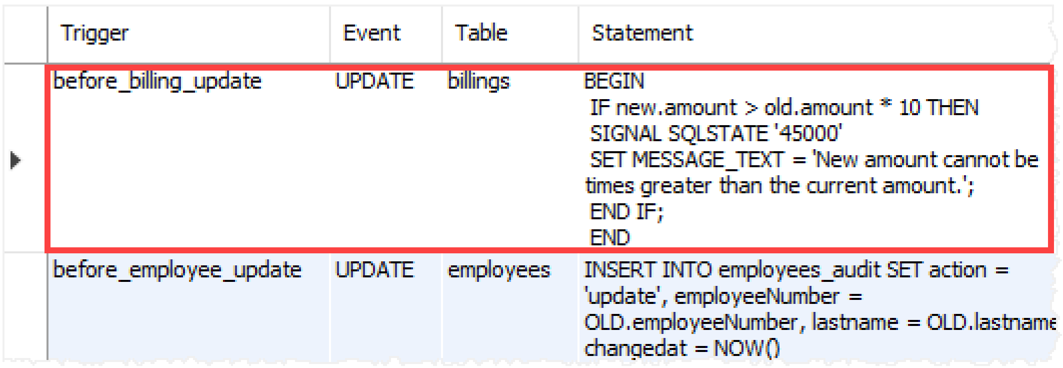
id	employeeNumber	lastname	changedat	action
1	1056	Patterson	2019-09-06 15:38:30	update

Drop Trigger:

DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name;

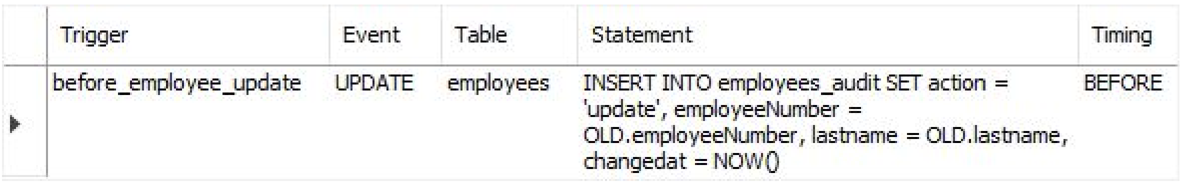
- If you drop a trigger that does not exist without using the IF EXISTS clause, MySQL issues an error. However, if you use the IF EXISTS clause, MySQL issues a NOTE instead.

- Specify the name of the schema to which the trigger belongs. If you skip the schema name, the statement will drop the trigger in the current database.



Trigger	Event	Table	Statement
before_billing_update	UPDATE	billings	BEGIN IF new.amount > old.amount * 10 THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'New amount cannot be times greater than the current amount.'; END IF; END
before_employee_update	UPDATE	employees	INSERT INTO employees_audit SET action = 'update', employeeNumber = OLD.employeeNumber, lastname = OLD.lastname changedat = NOW()

DROP TRIGGER before_billing_update;



Trigger	Event	Table	Statement	Timing
before_employee_update	UPDATE	employees	INSERT INTO employees_audit SET action = 'update', employeeNumber = OLD.employeeNumber, lastname = OLD.lastname, changedat = NOW()	BEFORE

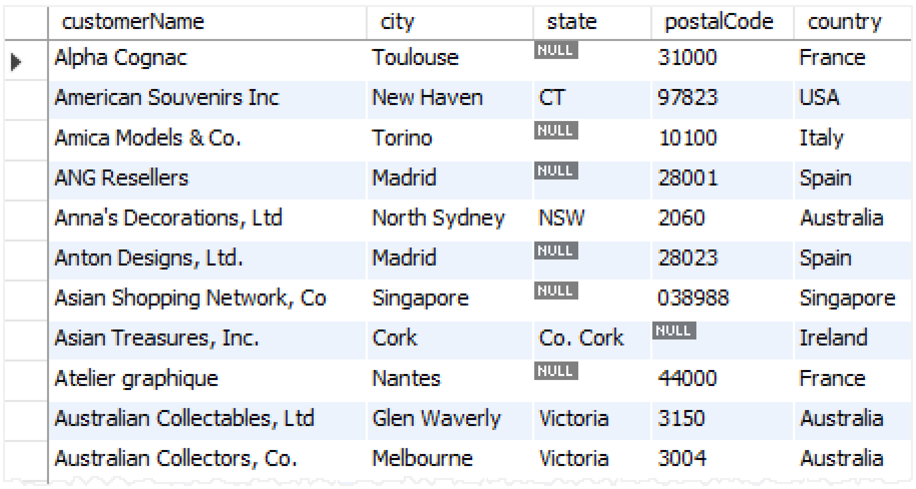
- Create a BEFORE INSERT trigger – Create a BEFORE INSERT trigger to maintain a summary table from another table.
- Create an AFTER INSERT trigger – Create an AFTER INSERT trigger to insert data into a table after inserting data into another table.
- Create a BEFORE UPDATE trigger – Create a BEFORE UPDATE trigger that validates data before it is updated to the table.
- Create an AFTER UPDATE trigger – Create an AFTER UPDATE trigger to log the changes of data in a table.
- Create a BEFORE DELETE trigger – Create a BEFORE DELETE trigger.
- Create an AFTER DELETE trigger – Create an AFTER DELETE trigger.

Stored Procedure

- A stored procedure is a set of declarative sql statements that can be invoked either directly or using triggers or events.

  The following SELECT statement returns all rows in the table customers

  SELECT customerName, city, state, postalCode, country FROM customers ORDER BY customerName;



customerName	city	state	postalCode	country
Alpha Cognac	Toulouse	NULL	31000	France
American Souvenirs Inc	New Haven	CT	97823	USA
Amica Models & Co.	Torino	NULL	10100	Italy
ANG Resellers	Madrid	NULL	28001	Spain
Anna's Decorations, Ltd	North Sydney	NSW	2060	Australia
Anton Designs, Ltd.	Madrid	NULL	28023	Spain
Asian Shopping Network, Co	Singapore	NULL	038988	Singapore
Asian Treasures, Inc.	Cork	Co. Cork	NULL	Ireland
Atelier graphique	Nantes	NULL	44000	France
Australian Collectables, Ltd	Glen Waverly	Victoria	3150	Australia
Australian Collectors, Co.	Melbourne	Victoria	3004	Australia

- If you want to save this query on the database server for execution later, one way to do it is to use a stored procedure.

  The following CREATE PROCEDURE statement creates a new stored procedure that wraps the query above:

  ```
  DELIMITER $$
  CREATE PROCEDURE GetCustomers()
  BEGIN
     SELECT customerName, city, state, postalCode, country FROM
     customers ORDER BY customerName;
  END$$
  DELIMITER ;
  ```

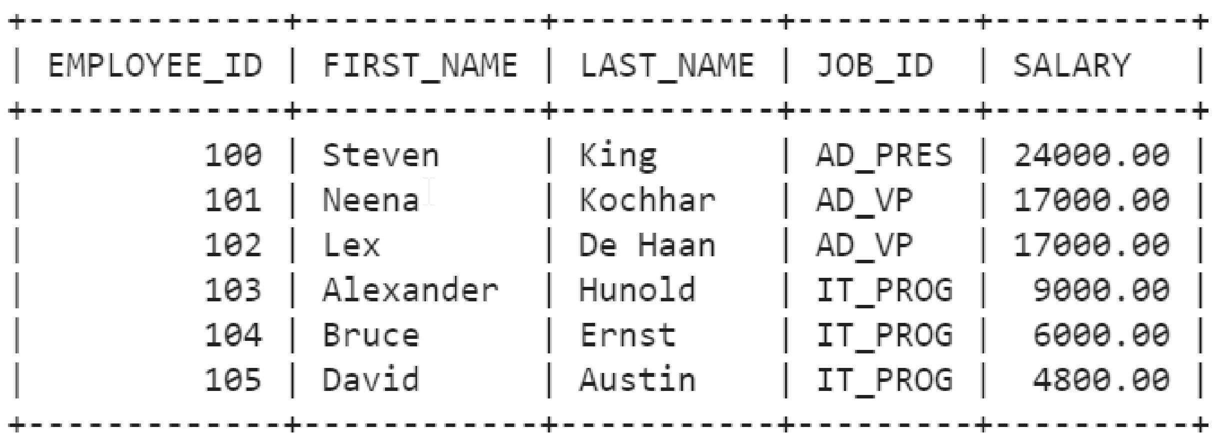
- Once you save the stored procedure, you can invoke it by using the CALL statement: CALL GetCustomers();

Triggers Examples

1. After Insert:

insert some information into log_emp_details table (which have three fields employee id and salary and edttime) every time, when an INSERT happens into emp_details table.

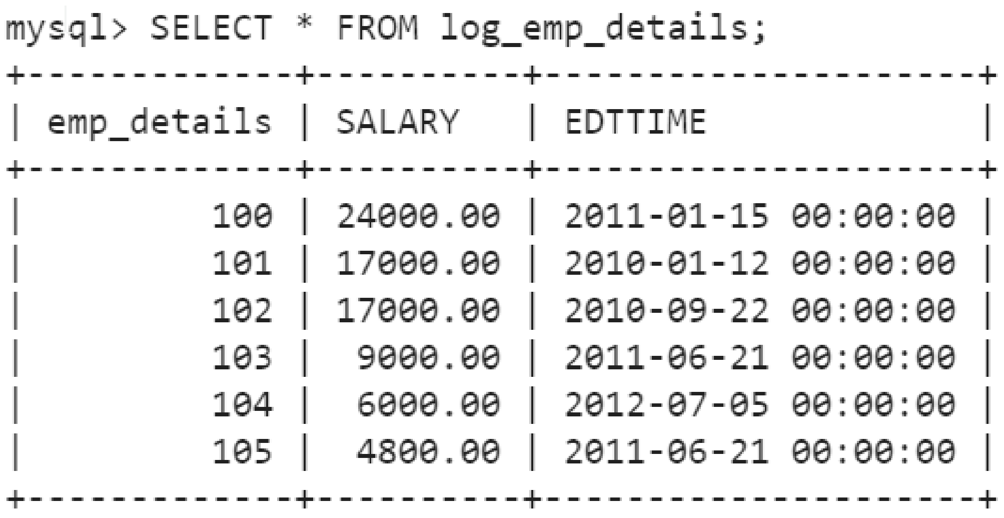
Emp_details table:



```
+-------------+------------+-----------+---------+----------+
| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | JOB_ID  | SALARY   |
+-------------+------------+-----------+---------+----------+
|         100 | Steven     | King      | AD_PRES | 24000.00 |
|         101 | Neena      | Kochhar   | AD_VP   | 17000.00 |
|         102 | Lex        | De Haan   | AD_VP   | 17000.00 |
|         103 | Alexander  | Hunold    | IT_PROG |  9000.00 |
|         104 | Bruce      | Ernst     | IT_PROG |  6000.00 |
|         105 | David      | Austin    | IT_PROG |  4800.00 |
+-------------+------------+-----------+---------+----------+
```

Trigger:

```
DELIMITER//
CREATE TRIGGER emp_details_AINS
AFTER INSERT ON emp_details
FOR EACH ROW
BEGIN
        INSERT INTO log_emp_details
        VALUES(NEW.employee_id, NEW.salary, NOW());
END//
DELIMITER;
```



```
mysql> SELECT * FROM log_emp_details;
+-------------+----------+---------------------+
| emp_details | SALARY   | EDTTIME             |
+-------------+----------+---------------------+
|         100 | 24000.00 | 2011-01-15 00:00:00 |
|         101 | 17000.00 | 2010-01-12 00:00:00 |
|         102 | 17000.00 | 2010-09-22 00:00:00 |
|         103 |  9000.00 | 2011-06-21 00:00:00 |
|         104 |  6000.00 | 2012-07-05 00:00:00 |
|         105 |  4800.00 | 2011-06-21 00:00:00 |
+-------------+----------+---------------------+
```

2. Before insert:

Before insert a new record in emp_details table, write a trigger to check the column value of FIRST_NAME, LAST_NAME, JOB_ID
- If there are any space(s) before or after the FIRST_NAME, LAST_NAME, (Hint: TRIM() function will remove those).
- The value of the JOB_ID convert to upper cases (hint: by UPPER() function).

FIRST_NAME - > '   Ana   ' has changed to 'Ana'
LAST_NAME - > '   King   ' has changed to 'King'
JOB_ID - > ' it_prog' has changed to 'IT_PROG'

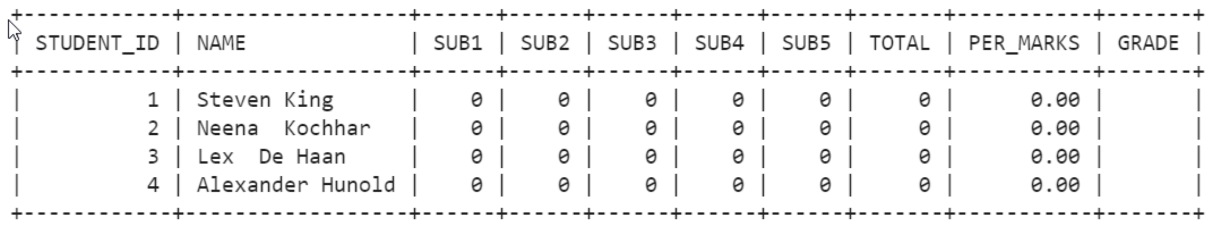
Trigger:

```
DELIMITER//
CREATE TRIGGER `emp_details_BINS`
BEFORE INSERT
ON emp_details FOR EACH ROW
BEGIN
        SET NEW.FIRST_NAME = TRIM(NEW.FIRST_NAME);
        SET NEW.LAST_NAME = TRIM(NEW.LAST_NAME);
        SET NEW.JOB_ID = UPPER(NEW.JOB_ID);
END//
DELIMITER;
```

3. Before update:

We have a table student_marks with 10 columns and 4 rows. There are data only in STUDENT_ID and NAME columns.

Student_marks table:



```
+------------+------------------+------+------+------+------+------+-------+-----------+-------+
| STUDENT_ID | NAME             | SUB1 | SUB2 | SUB3 | SUB4 | SUB5 | TOTAL | PER_MARKS | GRADE |
+------------+------------------+------+------+------+------+------+-------+-----------+-------+
|          1 | Steven King      |    0 |    0 |    0 |    0 |    0 |     0 |      0.00 |       |
|          2 | Neena  Kochhar   |    0 |    0 |    0 |    0 |    0 |     0 |      0.00 |       |
|          3 | Lex  De Haan     |    0 |    0 |    0 |    0 |    0 |     0 |      0.00 |       |
|          4 | Alexander Hunold |    0 |    0 |    0 |    0 |    0 |     0 |      0.00 |       |
+------------+------------------+------+------+------+------+------+-------+-----------+-------+
```

Now the exam is over and we have received all subject marks, now we will update the table, total marks of all subject, the percentage of total marks and grade will be automatically calculated. For this, the following conditions are assumed

Total Marks (stored in TOTAL column) : TOTAL = SUB1 + SUB2 + SUB3 + SUB4 + SUB5
Percentage of Marks (stored in PER_MARKS column) : PER_MARKS = (TOTAL)/5

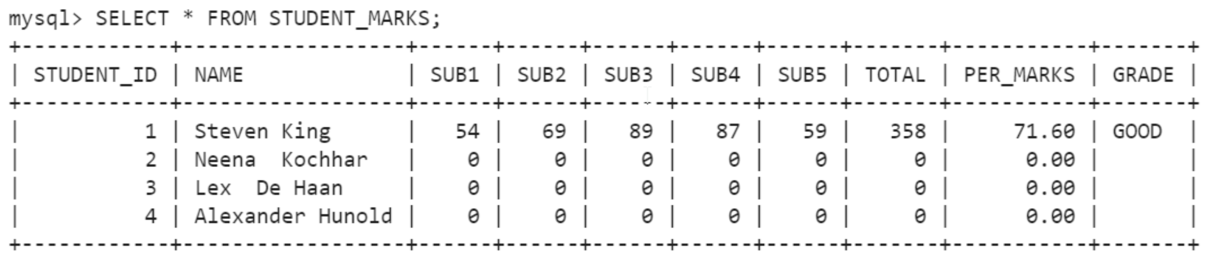
Grade (stored GRADE column) :
- If PER_MARKS>=90 -> 'EXCELLENT'
- If PER_MARKS>=75 AND PER_MARKS<90 -> 'VERY GOOD'
- If PER_MARKS>=60 AND PER_MARKS<75 -> 'GOOD'
- If PER_MARKS>=40 AND PER_MARKS<60 -> 'AVERAGE'
- If PER_MARKS<40-> 'NOT PROMOTED'

Trigger:

```
DELIMITER//
CREATE TRIGGER `student_marks_BUPD`
BEFORE UPDATE
ON student_marks FOR EACH ROW
BEGIN
        SET NEW.TOTAL = NEW.SUB1 + NEW.SUB2 + NEW.SUB3 + NEW.SUB4
+ NEW.SUB5;
        SET NEW.PER_MARKS = NEW.TOTAL/5;
        IF NEW.PER_MARKS >=90 THEN
                SET NEW.GRADE = 'EXCELLENT';
        ELSEIF NEW.PER_MARKS>=75 AND NEW.PER_MARKS<90 THEN
                SET NEW.GRADE = 'VERY GOOD';
        ELSEIF NEW.PER_MARKS>=60 AND NEW.PER_MARKS<75 THEN
                SET NEW.GRADE = 'GOOD';
        ELSEIF NEW.PER_MARKS>=40 AND NEW.PER_MARKS<60 THEN
                SET NEW.GRADE = 'AVERAGE';
        ELSE
                SET NEW.GRADE = 'NOT PROMOTED';
        END IF;
END//
DELIMITER;
```

Code:

```
UPDATE STUDENT_MARKS SET SUB1 = 54, SUB2 = 69, SUB3 = 89, SUB4 = 87,
SUB5 = 59 WHERE STUDENT_ID = 1
```



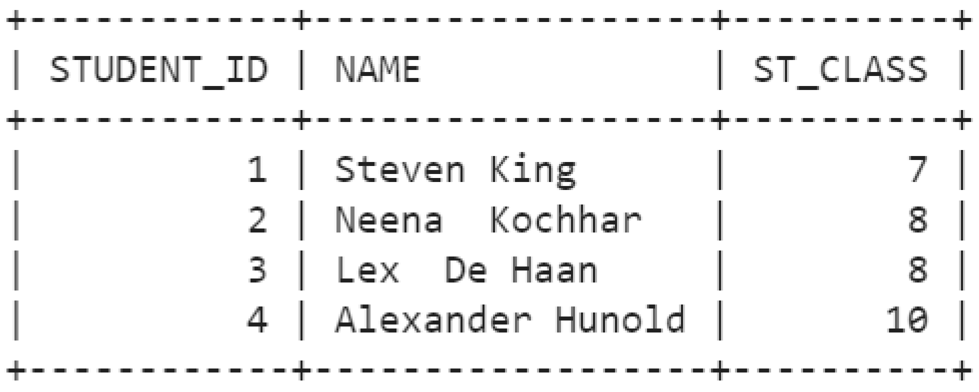
```
mysql> SELECT * FROM STUDENT_MARKS;
+------------+-----------------+------+------+------+------+------+-------+-----------+-------+
| STUDENT_ID | NAME            | SUB1 | SUB2 | SUB3 | SUB4 | SUB5 | TOTAL | PER_MARKS | GRADE |
+------------+-----------------+------+------+------+------+------+-------+-----------+-------+
|          1 | Steven King     |   54 |   69 |   89 |   87 |   59 |   358 |     71.60 | GOOD  |
|          2 | Neena  Kochhar  |    0 |    0 |    0 |    0 |    0 |     0 |      0.00 |       |
|          3 | Lex  De Haan    |    0 |    0 |    0 |    0 |    0 |     0 |      0.00 |       |
|          4 | Alexander Hunold|    0 |    0 |    0 |    0 |    0 |     0 |      0.00 |       |
+------------+-----------------+------+------+------+------+------+-------+-----------+-------+
```

4. After update:

two tables student_mast and stu_log.

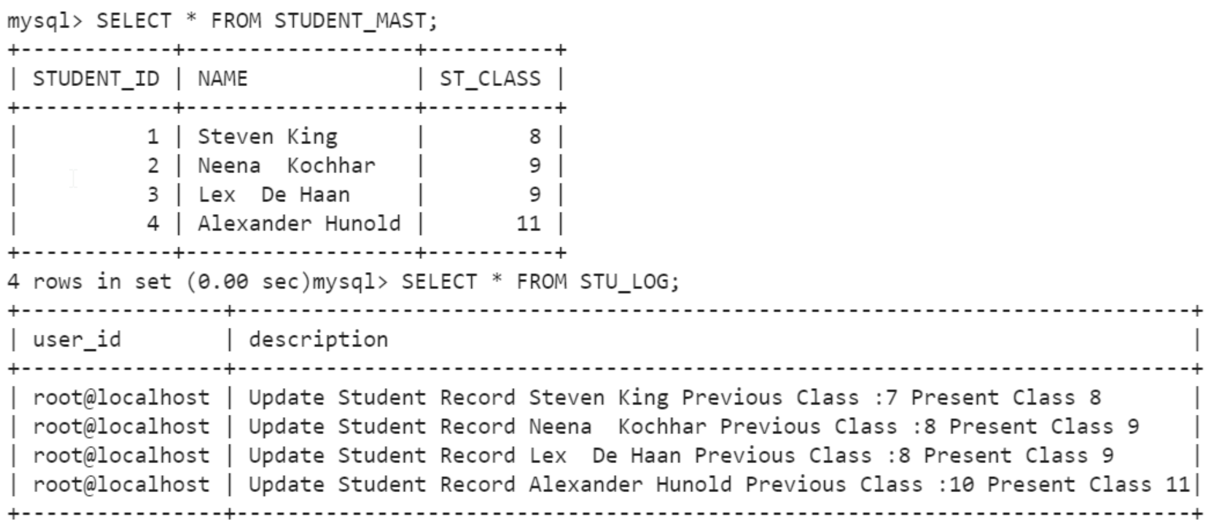
student_mast have three columns - STUDENT_ID, NAME, ST_CLASS.
stu_log table has two columns - user_id and description.

Student_mast table:



```
+------------+-----------------+----------+
| STUDENT_ID | NAME            | ST_CLASS |
+------------+-----------------+----------+
|          1 | Steven King     |        7 |
|          2 | Neena  Kochhar  |        8 |
|          3 | Lex  De Haan    |        8 |
|          4 | Alexander Hunold|       10 |
+------------+-----------------+----------+
```

Now let's promote all the students to next class i.e. 7 will be 8, 8 will be 9 and so on. After updating a single row in student_mast table, a new row will be inserted in stu_log table where we will store the current user id and a small description regarding the current update.



```
mysql> SELECT * FROM STUDENT_MAST;
+------------+-----------------+----------+
| STUDENT_ID | NAME            | ST_CLASS |
+------------+-----------------+----------+
|          1 | Steven King     |        8 |
|          2 | Neena  Kochhar  |        9 |
|          3 | Lex  De Haan    |        9 |
|          4 | Alexander Hunold|       11 |
+------------+-----------------+----------+
4 rows in set (0.00 sec)mysql> SELECT * FROM STU_LOG;
+----------------+---------------------------------------------------------------------------+
| user_id        | description                                                               |
+----------------+---------------------------------------------------------------------------+
| root@localhost | Update Student Record Steven King Previous Class :7 Present Class 8       |
| root@localhost | Update Student Record Neena  Kochhar Previous Class :8 Present Class 9    |
| root@localhost | Update Student Record Lex  De Haan Previous Class :8 Present Class 9      |
| root@localhost | Update Student Record Alexander Hunold Previous Class :10 Present Class 11|
+----------------+---------------------------------------------------------------------------+
```

Trigger:

```
DELIMITER//
CREATE TRIGGER student_mast_AUPD
AFTER UPDATE
ON student_mast FOR EACH ROW
BEGIN
        INSERT into stu_log VALUES (user(), CONCAT('Update Student Record ',
        OLD.NAME, ' Previous Class :', OLD.ST_CLASS, ' Present Class ',
        NEW.st_class));
END//
DELIMITER;
```

Code: `UPDATE STUDENT_MAST SET ST_CLASS = ST_CLASS + 1;`

5. After delete:

store some information in stu_log table after a delete operation happened on student_mast table

Trigger:

```
DELIMITER//
CREATE TRIGGER `student_mast_ADEL`
AFTER DELETE ON student_mast
FOR EACH ROW


BEGIN
        INSERT into stu_log VALUES (user(), CONCAT('Update Student Record ',
        OLD.NAME,' Clas :',OLD.ST_CLASS, '-> Deleted on ', NOW()));
END//
DELIMITER;
```

Now delete a student from STUDENT_MAST.

```
mysql> DELETE FROM STUDENT_MAST WHERE STUDENT_ID = 1;
mysql> SELECT * FROM STUDENT_MAST;
+------------+------------------+----------+
| STUDENT_ID | NAME             | ST_CLASS |
+------------+------------------+----------+
|          2 | Neena  Kochhar   |        9 |
|          3 | Lex  De Haan     |        9 |
|          4 | Alexander Hunold |       11 |
+------------+------------------+----------+
3 rows in set (0.00 sec)
mysql> SELECT * FROM STU_LOG;
+----------------+---------------------------------------------------------------------------+
| user_id        | description                                                               |
+----------------+---------------------------------------------------------------------------+
| root@localhost | Update Student RecordSteven King Previous Class :7 Present Class 8        |
| root@localhost | Update Student RecordNeena  Kochhar Previous Class :8 Present Class 9     |
| root@localhost | Update Student RecordLex  De Haan Previous Class :8 Present Class 9       |
| root@localhost | Update Student RecordAlexander Hunold Previous Class :10 Present Class 11 |
| root@localhost | Update Student Record Steven King Clas :8-> Deleted on 2013-07-16 15:35:30 |
+----------------+---------------------------------------------------------------------------+
```