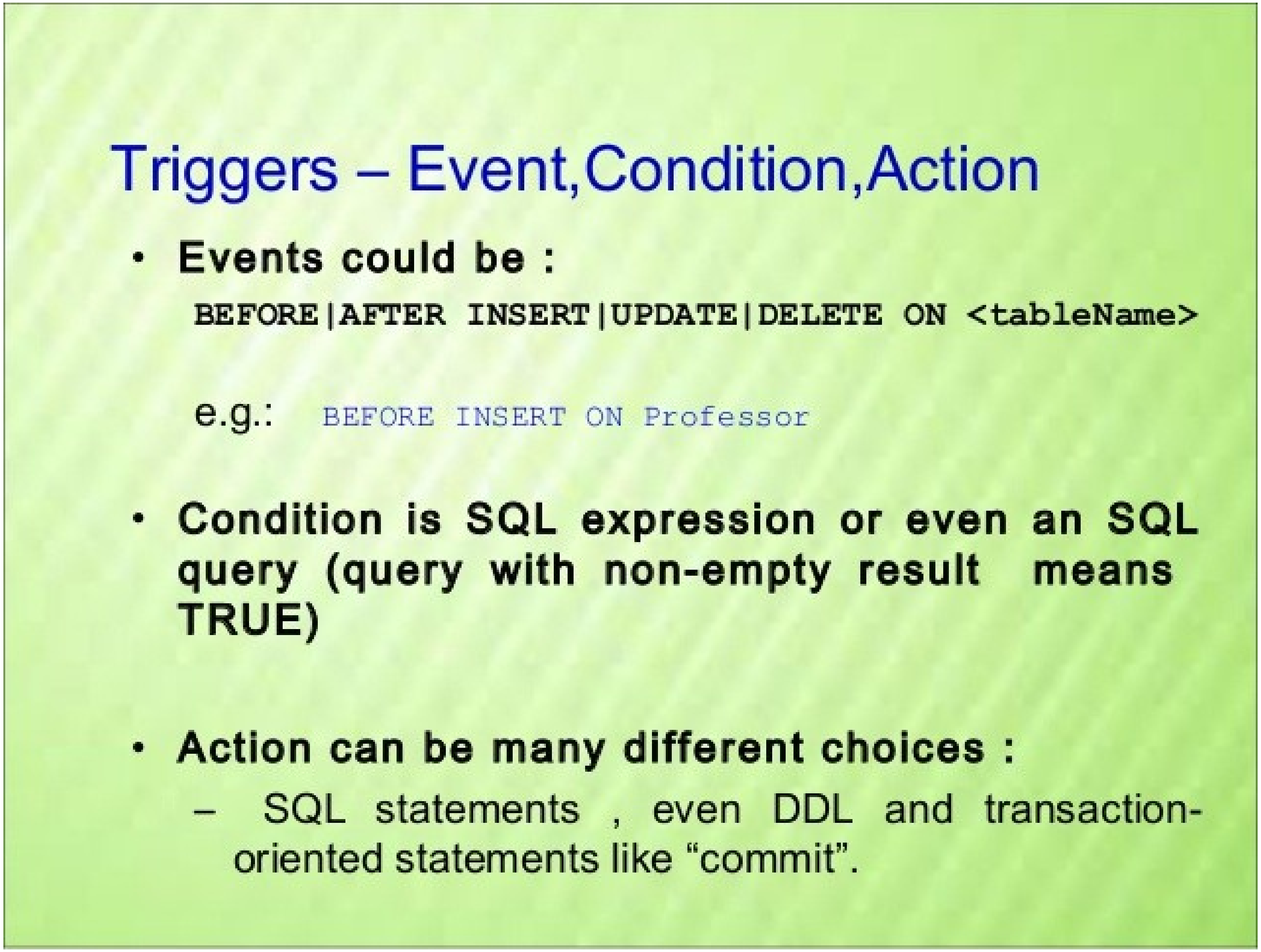
Trigger

- A procedure that starts automatically if specified changes occur to the DBMS

Triggers (Active database)

- **Three parts:**
  - Event (activates the trigger)
  - Condition (tests whether the triggers should run) [Optional]
  - Action (what happens if the trigger runs)
- **Semantics:**
  - When event occurs, and condition is satisfied, the action is performed.



Triggers – Event,Condition,Action

- **Events could be :**

  `BEFORE|AFTER INSERT|UPDATE|DELETE ON <tableName>`

  e.g.: `BEFORE INSERT ON Professor`

- **Condition is SQL expression or even an SQL query (query with non-empty result means TRUE)**

- **Action can be many different choices :**
  - SQL statements , even DDL and transaction-oriented statements like “commit”.

Defined by triggering transaction & level at which the trigger is executed

Types of Triggers

This section describes the different types of triggers:

- **Row level triggers**
  - Trigger once for "each row" in a transaction.
- **Statement level triggers**
  - Triggers execute once for "each transaction".
- **Before and after triggers**
  - Triggers can be executed immediately before and after INSERT, UPDATE and DELETE.

MySql Triggers

A SQL trigger is a set of SQL statements stored in the database catalog. A SQL trigger is executed or fired whenever an event associated with a table occurs e.g., insert, update or delete.

A SQL trigger is a special type of stored procedure. It is special because it is not called directly like a stored procedure. The main difference between a trigger and a stored procedure is that a trigger is called automatically when a data modification event is made against a table whereas a stored procedure must be called explicitly.

It is important to understand SQL trigger's advantages and disadvantages so that you can use it appropriately. In the following sections, we will discuss about the advantages and disadvantages of using SQL triggers.

Advantages of using SQL triggers

•SQL triggers provide an alternative way to check the integrity of data.

•SQL triggers can catch errors in business logic in the database layer.

•SQL triggers provide an alternative way to run scheduled tasks. By using SQL triggers, you don't have to wait to run the scheduled tasks because the triggers are invoked automatically *before* or *after* a change is made to the data in tables.

•SQL triggers are very useful to audit the changes of data in tables.

Disadvantages of using SQL triggers

•SQL triggers only can provide an extended validation and they cannot replace all the validations. Some simple validations have to be done in the application layer. For example, you can validate user's inputs in the client side by using JavaScript or in the server side using server side scripting languages such as JSP, PHP, ASP.NET, Perl, etc.

•SQL triggers are invoked and executed invisibly from client-applications therefore it is difficult to figure out what happen in the database layer.

•SQL triggers may increase the overhead of the database server.

Triggers or stored procedures? It is recommended that if you have no way to get the work done with stored procedure, think about SQL trigger.

Introduction to MySQL triggers

In MySQL, a trigger is a set of SQL statements that is invoked automatically when a change is made to the data on the associated table. A trigger can be defined to be invoked either before or after the data is changed by INSERT, UPDATE or DELETE statements. MySQL allows you to define maximum six triggers for each table.

BEFORE INSERT – activated before data is inserted into the table.

AFTER INSERT- activated after data is inserted into the table.

BEFORE UPDATE – activated before data in the table is updated.

AFTER UPDATE - activated after data in the table is updated.

BEFORE DELETE – activated before data is removed from the table.

AFTER DELETE – activated after data is removed from the table.

Introduction to MySQL triggers

When you use a statement that makes change to the table but does not use INSERT, DELETE or UPDATE statement, the trigger is not invoked. For example, the TRUNCATE statement removes the whole data of a table but does not invoke the trigger associated with that table.

There are some statements that use the INSERT statement behind the scenes such as REPLACE statement and LOAD DATA statement. If you use these statements, the corresponding triggers associated with the tables if available will be invoked.

Triggers defined for a table must have a unique name. You can have the same trigger name that defines for different tables but it is not recommended. In practice, the names of triggers follow the following naming convention:

1

(BEFORE | AFTER)_tableName_(INSERT| UPDATE | DELETE)

MySQL Triggers Storage

MySQL stores triggers in a data directory e.g., /data/classicmodels/ with the files namedtablename.TRG and triggername.TRN:

• The tablename.TRG file maps the trigger to the corresponding table.

• The triggername.TRN file contains the trigger definition.

You can back up the MySQL triggers by copying the trigger files to the backup folder. You can also backup the triggers using the *mysqldump* tool.

MySQL Trigger Limitations

MySQL triggers have all features in standard SQL however there are some limitations that you should know before using them in your applications.

MySQL triggers cannot:

- Use SHOW, LOAD DATA, LOAD TABLE, BACKUP DATABASE, RESTORE, FLUSH and RETURN statements.
- Use statements that commit or rollback implicitly or explicitly such as COMMIT, ROLLBACK,START TRANSACTION, LOCK/UNLOCK TABLES, ALTER, CREATE, DROP, RENAME, etc.
- Use prepared statements such as PREPARE, EXECUTE, etc.
- Use dynamic SQL statements.
- Call a stored procedure or stored function.

In this tutorial, we have shown you how triggers are implemented in MySQL. We also discussed about trigger's storage as well as trigger's limitations in MySQL.

MySQL Trigger Syntax

In order to create a trigger you use the CREATE TRIGGER statement. The following illustrates the syntax of the CREATE TRIGGER statement:-

```
CREATE TRIGGER trigger_name trigger_time trigger_event
 ON table_name
 FOR EACH ROW
 BEGIN
 ...
 END
```

MySQL Trigger Syntax

Let's examine the syntax above in more detail.

• You put the trigger name after the CREATE TRIGGER statement. The trigger name should follow the naming convention [trigger time]_[table name]_[trigger event], for examplebefore_employees_update.

• Trigger activation time can be BEFORE or AFTER. You must specify the activation time when you define a trigger. You use BEFORE keyword if you want to process action prior to the change is made on the table and AFTER if you need to process action after the change is made.

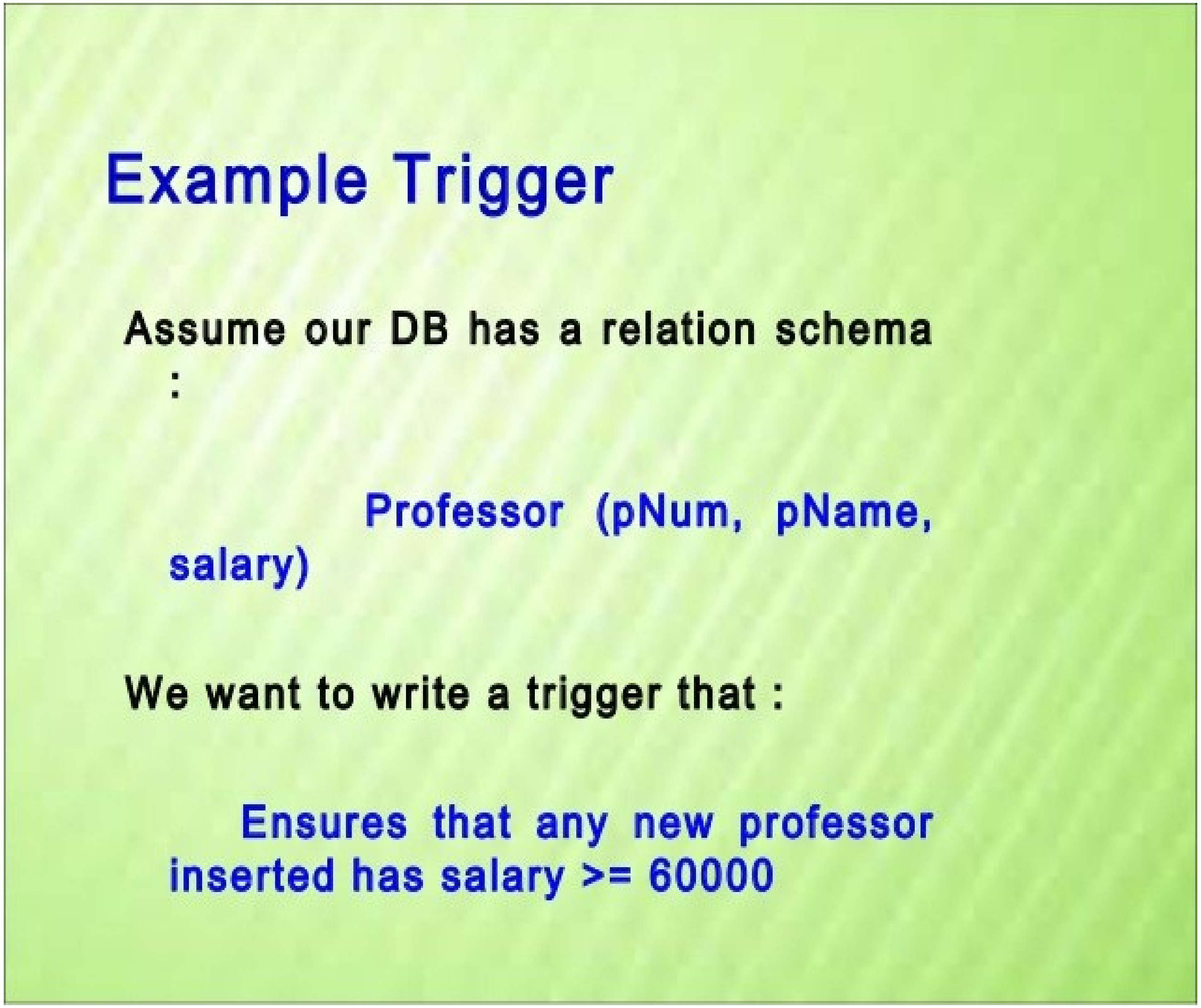
• Trigger event can be INSERT, UPDATE or DELETE. This event causes trigger to be invoked. Atrigger only can be invoked by one event. To define a trigger that is invoked by multiple events, you have to define multiple triggers, one for each event.

MySQL Trigger Syntax

- A trigger must be associated with a specific table. Without a table trigger would not exist therefore you have to specify the table name after the ON keyword.

- The SQL statements are placed between BEGIN and END block.

- The OLD and NEW keywords are very handy. The OLD keyword refers to the existing record before you change the data and the NEW keyword refers to the new row after you change the data.



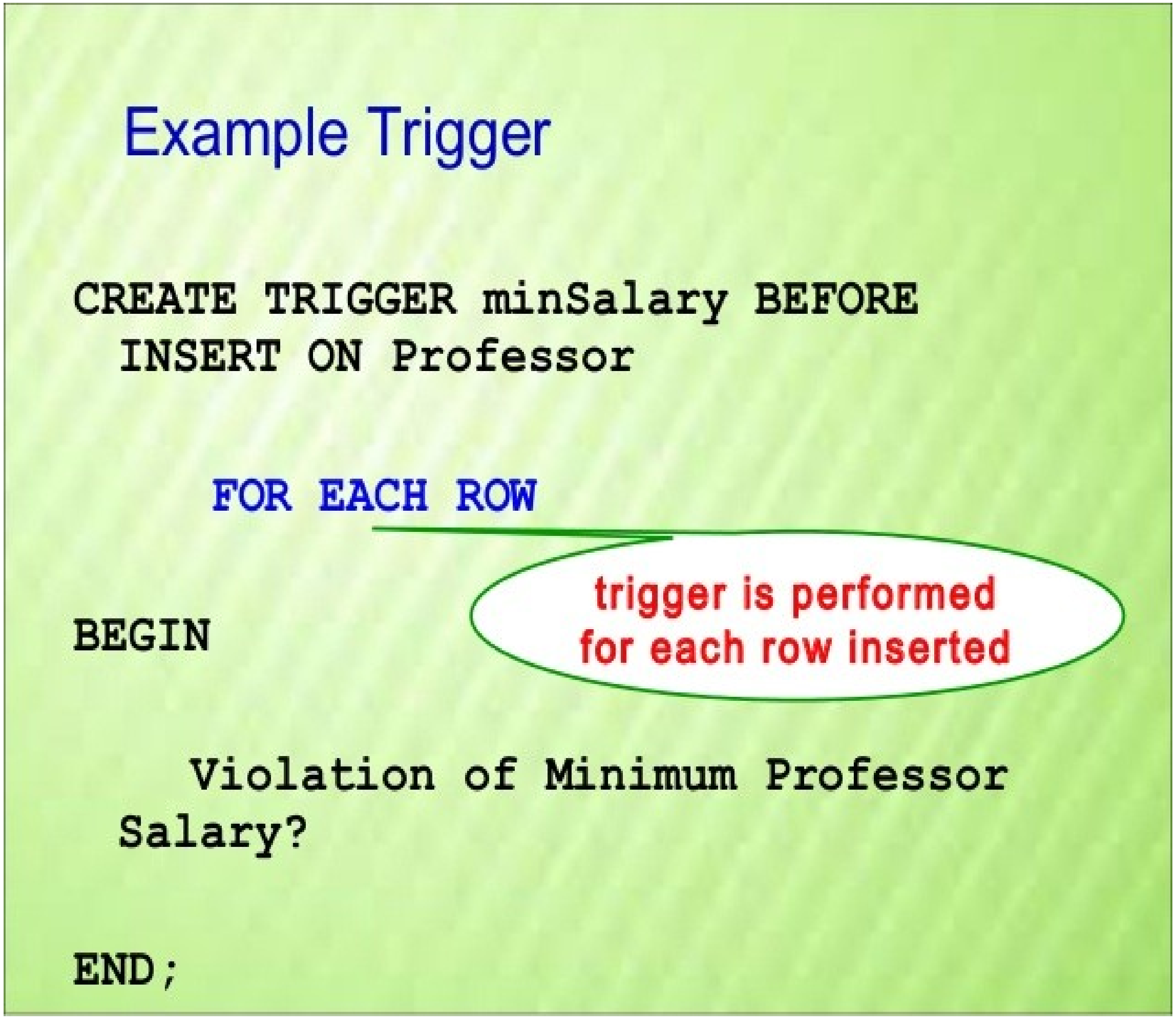
Example Trigger

Assume our DB has a relation schema :

Professor (pNum, pName, salary)

We want to write a trigger that :

Ensures that any new professor inserted has salary >= 60000



Example Trigger

```
CREATE TRIGGER minSalary BEFORE
  INSERT ON Professor

     FOR EACH ROW

BEGIN

     Violation of Minimum Professor
  Salary?

END;
```

trigger is performed
for each row inserted

Example Trigger

```
CREATE TRIGGER minSalary BEFORE INSERT ON
  Professor

     FOR EACH ROW

BEGIN

  IF (:new.salary < 60000)
     THEN RAISE_APPLICATION_ERROR (-20004,
               'Violation of Minimum
  Professor Salary');
  END IF;

END;
```

Details of Trigger Example

- **BEFORE INSERT ON Professor**
  - This trigger is checked before the tuple is inserted
- **FOR EACH ROW**
  - specifies that trigger is performed for each row inserted
- **:new**
  - refers to the new tuple inserted
- **If (:new.salary < 60000)**
  - then an application error is raised and hence the row is not inserted; otherwise the row is inserted.
- **Use error code: -20004;**
  - this is in the valid range

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

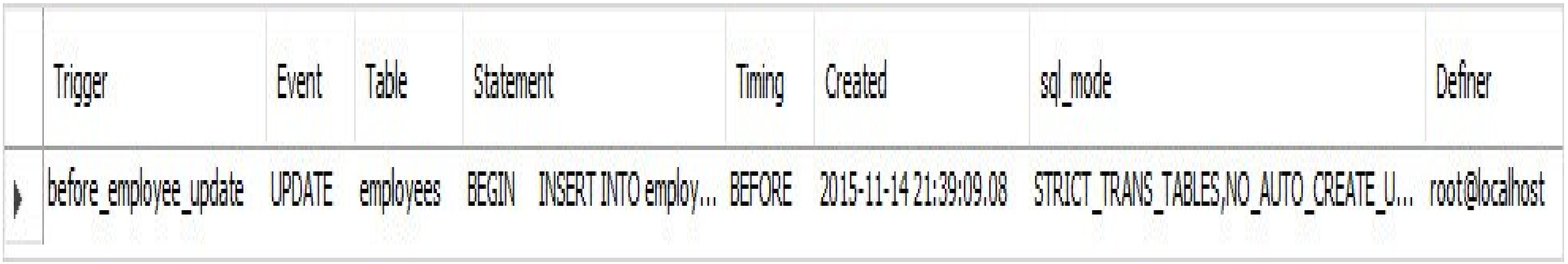
```
DELIMITER $$
CREATE TRIGGER before_employee_update
    BEFORE UPDATE ON employees
    FOR EACH ROW
BEGIN
    INSERT INTO employees_audit
    SET action = 'update',
         employeeNumber = OLD.employeeNumber,
        lastname = OLD.lastname,
        changedat = NOW();
END$$
DELIMITER ;
```

- Inside the body of the trigger, we used the OLD keyword to access employeeNumber and lastname column of the row affected by the trigger.

- Notice that in a trigger defined for INSERT, you can use NEW keyword only. You cannot use the OLD keyword. However, in the trigger defined for DELETE, there is no new row so you can use the OLD keyword only. In the UPDATE trigger, OLD refers to the row before it is updated and NEW refers to the row after it is updated.

- Then, to view all triggers in the current database, you use SHOW TRIGGERS statement as follows

```
SHOW TRIGGERS;
```



Trigger	Event	Table	Statement	Timing	Created	sql_mode	Definer
before_employee_update	UPDATE	employees	BEGIN INSERT INTO employ...	BEFORE	2015-11-14 21:39:09.08	STRICT_TRANS_TABLES,NO_AUTO_CREATE_U...	root@localhost