





General enterprise constraints are often too complicated to be enforced using foreign keys and CHECK.

Triggers are used to perform actions when a defined event occurs. They are standardized, but also have been present in various database systems for some time.

Triggers allow general constraints to be enforced in response to database events.





Triggers are also called event-condition-action (ECA) rules.

When an *event* occurs (such as inserting a tuple) that satisfies a given *condition* (any SQL boolean-valued expression) an *action* is performed (which is any sequence of SQL statements).

Triggers provide a general way for detecting events and responding to them.





Consider this situation where triggers are useful.

The workson relation has a foreign key to emp (eno). If a user inserts a record in workson and the employee does not exist, the insert fails.

However with triggers, we can accept the insertion into workson and then create a new record in emp so that the foreign key constraint is not violated.





CREATE TRIGGER insertWorksOn

```
BEFORE INSERT ON WorksOn
REFERENCING NEW ROW AS NewWO
                                 Condition
FOR EACH ROW
WHEN (NewWO.eno NOT IN
          (SELECT eno FROM emp))
       INTO
             Emp (eno)
                                  Action
     VALUES (NewWO.eno);
```

Triggers Standard Syntax



```
CREATE TRIGGER <name>
   BEFORE | AFTER | INSTEAD OF <events>
   [<referencing clause>]
   [FOR EACH ROW]
   [WHEN (<condition>)]
   <action>
```

Notes:

- BEFORE, AFTER, INSTEAD OF indicate when a trigger is executed.
- <events> is the events that the trigger will be executed for. It will be one of these events: INSERT ON R
 DELETE ON R

UPDATE [OF A1, A2, ..., An] on R





There are two types of triggers:

- row-level triggers that are executed for each row that is updated, deleted, or inserted.
- **statement-level triggers** that are only executed once per statement regardless of how many tuples are affected.

Inserting the clause FOR EACH ROW indicates a row-level trigger (the default is a statement-level trigger).



Triggers Syntax - Referencing

The referencing clause allows you to assign names to the row or table being affected by the triggered event:

- INSERT statements imply a new tuple (for row-level) or new set of tuples (for statement-level).
- DELETE implies an old tuple or table.
- UPDATE implies both.

These tuples or tables can be referred to using the syntax:

```
[NEW OLD] [TUPLE TABLE] AS <name>
```

Example: Statement-level trigger on an update:

```
REFERENCING OLD TABLE AS oldTbl
NEW TABLE AS newTbl
```



Triggers Syntax - Condition

The condition is used to decide when the trigger action is performed.

• For example, even though an insert event may occur, you may only want to execute the trigger if a certain condition holds.

Any SQL boolean-valued condition can be used. The condition clause is evaluated before or after the triggering event, depending on whether BEFORE or AFTER is used in the event.





The action is performed when the event occurs and the condition is satisfied.

The action is one or more SQL statements.

If more than one SQL statement is executed, they should be in a BEGIN .. END block.





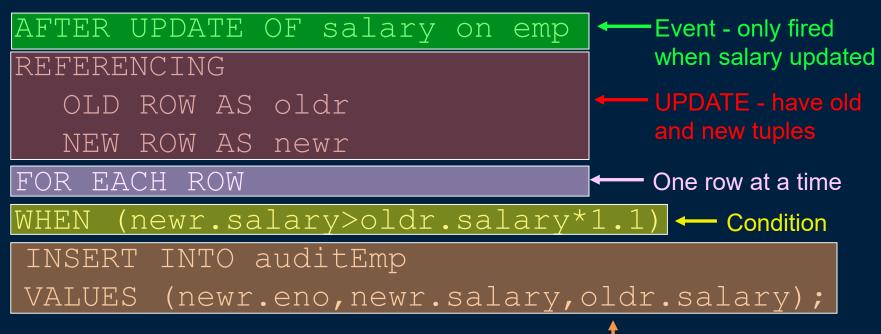
Consider a trigger that is used to monitor employee salary increases.

If an employee's salary is increased by more than 10%, we want to add that employee to a relation auditEmp (eno, newSalary, oldSalary) that is used by the accounting department to verify that the employee is not stealing from the company.

Example Trigger #2



CREATE TRIGGER cheating Employee



Action - add new row to auditEmp





One interesting use of triggers is to allow updates on views that are not normally updatable.

This is done by using the INSTEAD OF clause. When an update is executed, instead of executing the update command given by the user, the trigger executes its own SQL statements.

For instance, this may allow it to perform inserts on many tables that were combined to form a single view.





Question: True or False: A standard trigger can be used to perform an action in response to a SELECT query.

A) true

B) false





The syntax presented is for the SQL standard. Each database has its own syntax and level of support for triggers.

Feature	MySQL	SQL Server	PostgreSQL
Statement-level triggers	No	Yes	Yes
Row-level triggers	Yes	No	Yes
OLD/NEW Table	No	Yes. Inserted and deleted tables.	No
OLD/NEW Row	Yes. Use New and Old as names.	No	Yes. Use New and Old as names.
INSTEAD OF trigger	No	Yes, with limits	Yes, only on views
WHEN clause	No		Yes



Example Trigger in MySQL Syntax

```
CREATE TRIGGER cheating Employee
   AFTER UPDATE ON Emp
   FOR EACH ROW
   BEGIN
    IF NEW.salary > OLD.salary*1.1 THEN
      INSERT INTO auditEmp
       VALUES (NEW.eno, NEW.salary, OLD.salary);
    END IF;
   END;
```



Example Trigger in SQL Server Syntax

```
CREATE TRIGGER cheatingEmployee ON Emp

AFTER UPDATE AS

BEGIN

INSERT INTO auditEmp

SELECT d.eno, newsal, oldsal FROM

deleted d JOIN inserted i ON d.eno = i.eno

WHERE i.salary > d.salary*1.1

END;
```





Write triggers for these cases:

- 1) When an employee is added to the emp table in department 'D1' with no supervisor make their supervisor 'E8'.
- 2) Write the same trigger as a statement-level trigger.
- 3) Write a trigger that deletes all tuples in the workson relation for an employee when the employee is deleted from the Emp relation.

```
emp (eno, ename, bdate, title, salary, supereno, dno)
proj (pno, pname, budget, dno)
dept (dno, dname, mgreno)
workson (eno, pno, resp, hours)
```





More complicated constraints can be enforced in the database using triggers.

Triggers allow for efficient enforcement of general constraints by specifying specific events to monitor and actions to take.

Objectives



- Create triggers from high-level descriptions
- Explain the difference between row level and statement level triggers
- List the events that triggers are used for
- Explain how triggers can be used for views

