Chapter 7
Triggers

A trigger is an element of the database schema that has the following structure:

On event If precondition Then action

where event is a request for the execution of a particular database operation, e.g., add in a row in the table; precondition is a boolean expression as when this trigger can be fired; and action is a statement of what needs done when this trigger is fired.

When a student registers for a course (event), if the class is already full (the associated condition), a message should be displayed and this insertion will be declined (action). On the other hand, if it is not full yet, the insertion will go ahead.
Where is a trigger used?

1. It can be used to maintain constraints. The most common form of a trigger in this kind uses the On Delete and On Update clauses, which are attached to foreign key clauses, so that, e.g., when something is to be deleted, but some other rows are still referring to it, a trigger can be fired to prevent this deletion from happening.

2. A business rule is a concise statement about a basic principle underlying a business process.

For example, a trigger could state that if an international money transfer is made into a client’s account, and it is not suspicious, then an Emil message should be sent to notify the client. But, if it is suspicious, some agency will be contacted.
3. complex physical objects are monitored by sensor networks, which will add in the measurement into a database. Since such insertion is a database event, we can use a trigger to take some action when needed.

For example, if too much $CO_2$ is detected when the record is being inserted, a ventilation system can be turned on as a result.

4. A trigger can also be used to update a view, e.g., whenever a change is made to the base tables.

5. When exceptions need to be handled in database applications, triggers are ideal.

For example, when divided by zero.
Issues with triggers

1. There are quite a few types of triggers, which might be useful for different kinds of applications. Which one(s) should be used?

2. At a certain point, several triggers are ready to be fired. Which one should be picked up first? and more generally, what is the order of the execution of those triggers?

3. Execution of one trigger might cause the execution of other triggers, sort of a chain of reaction. To prevent an infinite loop, DBMS has to put a limit on the length of such a chain.

When such a limit is reached, all the changes by the original update and those already fired triggers will be rolled back.
Trigger consideration

A trigger is *activated* when the triggering event is requested. The *consideration* of a trigger refers to *when* the precondition is checked after a trigger is activated.

If the precondition is checked out to be true, then the trigger should be fired, i.e., the associated action should be taken. But, moments later, this precondition could become false, maybe caused by the update made by this very requested event; or something else.

Thus, *when the precondition is checked* becomes an issue.
An example

Consider the following trigger:

On inserting a row in CourseRegistration table
If over course capacity
Then abort registration transaction

When a student attempts to add a course in the registration table, the course might be full, when this insertion should be rejected.

But, it might be the case that a bit later, another student is withdrawing from this course, or the capacity is increased. Thus, if the condition is checked at a later time, this insertion might be doable.
It all depends....

Apparently, in this case, deferring the consideration of this update might be a good idea. Sort of putting him into a waiting list. Later on, when a student withdraws, the associated action could be to add in the top on the waiting list.

However, if the database is to monitor the pressure in a nuclear power plant and the trigger event is pressure increase while the precondition is that the pressure not to exceed a preset level, then an immediate check of the condition is definitely mandatory.

Hence, there are at least two approaches as when the precondition should be checked: immediate or deferred until the triggering transaction is committed.
Trigger execution

If trigger consideration is deferred, its execution is necessarily deferred as well until the end of the triggering transaction.

On the other hand, if the precondition is immediately checked, we can either execute the trigger immediately after the check of the precondition; or we can defer execution of the trigger until the end of the triggering transaction.

It again depends on the application. For the nuclear power plant case, it should be done immediately.
More possibilities

Normally, the action as defined for the trigger is executed after the triggering event, which causes the firing of the trigger, and is referred to as an after trigger. But, it can also be executed before the event, thus before trigger.

We can also do something else, instead of the event; i.e., instead of the triggering event, another action will be taken instead, thus an instead trigger.

As an example, when a tuple is inserted into a view (a triggering event), a trigger can be executed to insert the row into the base table(s) of the view (the action of the trigger) instead of inserting into the view itself.
Triggers in SQL

In SQL 99, an event can be either Delete, Insert, or Update.

Any conditions allowed in the Where part can be used as a triggering precondition.

A triggering action can be Delete, Insert, Update, Rollback, a Signal statement, or even a program.

SQL follows the ordered resolution, and always consider triggers as immediate.
Before triggers

Execution of a trigger can be specified as either before or after the triggering event. All Before triggers execute, often a test, entirely before the triggering events, without changing the database.

Another interesting point is that we can reference the old table and the new table using the Referencing Old As and Referencing New As clauses, which are the pre-update and post-update data as contained in the affected table.

More specifically, the New As and Old As specify only those tuples that are affected. Thus, e.g., for an insertion event, New As refers to the tuple being inserted.
An example

Assume that our registration database includes a CrsLimits(CrsCode, Semester, Limit) table, we could have the following trigger that enforces the course limit by monitoring insertions into the Transcript table.

Create Trigger RoomCapacityCheck
   Before Insert On Transcript
       Referencing New As N
   For each row
       When ((Select Count(T.StudId) From Transcript T
           Where T.CrsCode=N.CrsCode A
           And T.Semester=N.Semester)
           >=
           Select.Limit From CrsLimits L
           Where L.CrsCode=N.CrsCode
           And L.Semester=N.Semester))
           Rollback

Notice that an insertion can add in several tuples. Thus, this trigger will be fired for each row added.
A MySQL example

Trigger got into MySQL, starting with version 5.0. Assume we want to see the running total of an account, which is a table

mysql> CREATE TABLE account (  
    -> acct_num INT,  
    -> amount DECIMAL(10,2)  
    -> );  
Query OK, 0 rows affected (0.09 sec)

We can create a user variable as follows:

mysql> Set @sum=0;  
Query OK, 0 rows affected (0.00 sec)
We can then create a before trigger `ins_sum`, which adds up the value into a variable `sum`.

```sql
mysql> CREATE TRIGGER ins_sum
  -> BEFORE INSERT ON account
  -> FOR EACH ROW
  -> SET @sum = @sum + NEW.amount;
Query OK, 0 rows affected (0.08 sec)
```

For example,

```sql
mysql> INSERT INTO account
  -> VALUES(137,14.98),(141,1937.50),(97,-100.00);
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

```sql
mysql> SELECT @sum AS 'Total amount inserted';
+-----------------------+
| Total amount inserted |
+-----------------------+
| 1852.48               |
+-----------------------+
1 row in set (0.00 sec)
```

Check out the MySQL site for more examples.
After triggers

All *After triggers* execute entirely after the triggering event has applied all its changes to the database. They are allowed to change the database, and thus can cause other triggers to fire as well.

Essentially implementing a conditional structure, an *after trigger* can be perceived as an extension of the application logic, which takes care of various events automatically, thus relieving programmers of the need to code all these pieces for all cases.
An example

The following caps any salary change by 5%, assuming the Employee table has an attribute of Salary.

Create Trigger LimitSalaryRaise
   After Update Of Salary On Employee
   Referencing Old as O
       New As N
   For each row
   When (N.Salary-0.Salary > 0.05*0.Salary)
       Update Employee
       Set Salary =1.05*0.Salary
       Where Id=0.Id

Notice that the tuples 0 and N always refer to the same tuple that is being updated.