UNIT II SPATIAL AND TEMPORAL DATABASES 9

Active Databases Model – Design and Implementation Issues - Temporal Databases - Temporal Querying - Spatial Databases: Spatial Data Types, Spatial Operators and Queries – Spatial Indexing and Mining – Applications -– Mobile Databases: Location and Handoff Management, Mobile Transaction Models – Deductive Databases - Multimedia Databases.

ACTIVE DATABASES MODEL

A trigger is a procedure which is automatically invoked by the DBMS in response to changes to the database, and is specified by the database administrator (DBA). A database with a set of associated triggers is generally called an active database.

Parts of trigger

A triggers description contains three parts, which are as follows

- The event(s) that triggers the rule: These events are usually database update operations that are explicitly applied to the database. However, in the general model, they could also be temporal events2 or other kinds of external events
- The condition that determines whether the rule action should be executed: Once the triggering event has occurred, an optional condition may be evaluated. If no condition is specified, the action will be executed once the event occurs. If a condition is specified, it is first evaluated, and only if it evaluates to true will the rule action be executed
- The action to be taken: The action is usually a sequence of SQL statements, but it could also be a database transaction or an external program that will be automatically executed.

Use of trigger

Triggers may be used for any of the following reasons −

- To implement any complex business rule, that cannot be implemented using integrity constraints.
- Triggers will be used to audit the process. For example, to keep track of changes made to a table.
- Trigger is used to perform automatic action when another concerned action takes place.

Types of triggers

The different types of triggers are explained below −

- **Statement level trigger** − It is fired only once for DML statements irrespective of the number of rows affected by the statement. Statement-level triggers are the default type of trigger.

- **Before-triggers** − At the time of defining a trigger we can specify whether the trigger is to be fired before a command like INSERT, DELETE, or UPDATE is executed or after the command is executed. Before triggers are automatically used to check the validity of data before the action is performed. For instance, we can use before trigger to prevent deletion of rows if deletion should not be allowed in a given case.
- **After-triggers** − It is used after the triggering action is completed. For example, if the trigger is associated with the INSERT command then it is fired after the row is inserted into the table.
- **Row-level triggers** − It is fired for each row that is affected by DML command. For example, if an UPDATE command updates 150 rows then a row-level trigger is fired 150 times whereas a statement-level trigger is fired only for once.

Create database trigger

To create a database trigger, we use the CREATE TRIGGER command. The details to be given at the time of creating a trigger are as follows −

- Name of the trigger.
- Table to be associated with.
- When trigger is to be fired: before or after.
- Command that invokes the trigger- UPDATE, DELETE, or INSERT.
- Whether row-level triggers or not.
- Condition to filter rows.
- PL/SQL block is to be executed when trigger is fired.

The syntax to create database trigger is as follows:

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE|AFTER}
{DELETE|INSERT|UPDATE[OF COLUMNS]} ON table
[FOR EACH ROW {WHEN condition]]
[REFERENCE [OLD AS old] [NEW AS new]]
BEGIN
PL/SQL BLOCK
END.
```

DESIGN AND IMPLEMENTATION ISSUES FOR ACTIVE DATABASES

In this section, we discuss some additional issues concerning how rules are designed and implemented.

The first issue concerns activation, deactivation, and grouping of rules. In addition to creating rules, an active database system should allow users to activate, deactivate, and drop rules by referring to their rule names. A deactivated rule will not be triggered by the triggering event. This feature allows users to selectively deactivate rules for certain periods of time when they are not needed. The activate command will make the rule active again. The drop command deletes the rule from the system. Another option is to group rules into named rule sets, so the whole set of rules can be activated, deactivated, or dropped. It is also useful to have a command that can trigger a rule or rule set via an explicit PROCESS RULES command issued by the user.

The second issue concerns whether the triggered action should be executed before, after, instead of, or concurrently with the triggering event. **A before trigger** executes the trigger before executing the event that caused the trigger. It can be used in appli- cations such as checking for constraint violations. **An after trigger** executes the trig- ger after executing the event, and it can be used in applications such as maintaining derived data and monitoring for specific events and conditions. **An instead of trigger** executes the trigger instead of executing the event, and it can be used in applications such as executing corresponding updates on base relations in response to an event that is an update of a view.

A related issue is whether the action being executed should be considered as a separate transaction or whether it should be part of the same transaction that triggered the rule. We will try to categorize the various options. It is important to note that not all options may be available for a particular active database system.

In fact, most commercial systems are limited to one or two of the options that we will now discuss. Let us assume that the triggering event occurs as part of a transaction execution. We should first consider the various options for how the triggering event is related to the evaluation of the rule's condition.

The rule condition evaluation is also known as rule consideration , since the action is to be executed only after considering whether the condition evaluates to true or false. There are three main possibilities for rule consideration:

1.Immediate consideration. The condition is evaluated as part of the same transaction as the triggering event, and is evaluated immediately.

This case can be further categorized into three options:

- Evaluate the condition before executing the triggering event.
- Evaluate the condition after executing the triggering event.
- Evaluate the condition instead of executing the triggering event.

2. Deferred consideration. The condition is evaluated at the end of the trans- action that includes the triggering event. In this case, there could be many

3. Detached consideration. The condition is evaluated as a separate transaction, spawned from the triggering transaction.

The next set of options concerns the relationship between evaluating the rule condition and executing the rule action. Here, again, three options are possible: immediate , deferred, or detached execution. Most active systems use the first option.

That is, as soon as the condition is evaluated, if it returns true, the action is immediately executed. The Oracle system uses the immediate consideration model, but it allows the user to specify for each rule whether the before or after option is to be used with immediate condition evaluation. It also uses the immediate execution model.

The STARBURST system uses the deferred consideration option, meaning that all rules triggered by a transaction wait until the triggering transaction reaches its end and issues its COMMIT WORK command before the rule conditions are evaluated.

Another issue concerning active database rules is the distinction between row-level rules and statement-level rules. Because SQL update statements (which act as triggering events) can specify a set of tuples, one has to distinguish between whether the rule should be considered once for the whole statement or whether it should be considered separately for each row (that is, tuple) affected by the statement.