1. **Introduction**

Depuis de nombreuses années le besoin de pouvoir faire des mises à jour automatique a toujours été un but prépondérant dans la conception des bases de données. Ce dans ce cadre que les déclencheurs ont été créés

1. **Définition et Fonctionnement d’un trigger**

Un trigger est un programme résident associé à un évènement particulier. Une fois que cet évènement se produit, le trigger se lance et ses instructions sont exécutées. Un trigger est comme une procédure stockée dans le sens où les deux sont des blocs d’instructions déjà stockés et dont on fait appel lorsque c’est nécessaire. Néanmoins, il diffère des procédures stockées par le fait qu’il est lancé lorsqu’un événement se produit alors que la procédure stockée elle, est lancée lorsqu’une commande l’appelant est passée.

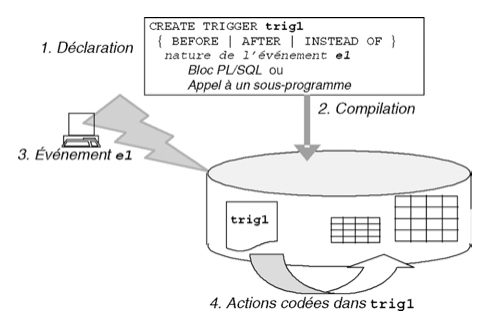
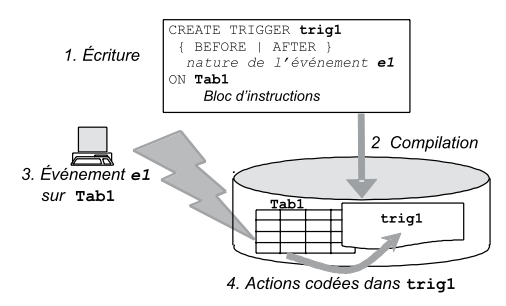
Les événements déclencheurs peuvent être :

* Une instruction INSERT, UPDATE, ou DELETE sur une table (ou vue). On parle de **déclencheurs LMD** ;
* Une instruction CREATE, ALTER, ou DROP sur un objet (table, vue, index, séquence etc). On parle de **déclencheurs LDD** ;
* Le démarrage ou l’arrêt de la base (*startup* ou *shutdown*), une erreur spécifique *(not found*, *duplicate key*, etc.), une connexion ou une déconnexion d’un utilisateur. On parle de **déclencheurs d’instances**.

En pratique, dans MySQL, seuls les événements de la première catégorie sont pris en compte

Une fois qu’un trigger a été écrit, il est compilé et stocké dans la base de données. Une fois que le trigger est compilé et stocké, il attend un événement pour être lancé. A chaque fois qu’un évènement caractérisant le trigger se produit, le trigger sera lancé et ses instructions seront exécutées, comme une procédure stockée.

* Une petite différence existe néanmoins dans l’implémentation des triggers dans les SGBD Oracle et MySQL. Dans Oracle, c’est le trigger qui appelle la table sur laquelle il est placé afin de pouvoir s’exécuter tandis que sur MySQL c’est un événement sur la table qui appelle le trigger pour s’exécuter.



***MySQL Oracle***

*Fig : Mécanisme des déclencheurs*

1. **Avantages et Inconvénients des triggers**

Afin de répondre à la question « pourquoi le trigger ?», nous allons poser le problème suivant : «**Lors de la maintenance d’une base de données en cohérence avec la logique métier, quand est-il préférable d’utiliser un trigger plutôt qu’effectuer les vérifications nécessaires dans la partie application ? »** Pour y répondre, nous allons examiner les avantages et inconvénients de l’utilisation des triggers.

**a - Avantages des triggers :**

* Les triggers **fournissent un gain en temps**: l’exécution des triggers s’effectuant directement au niveau de la base données, ils limitent les échanges entre le serveur applicatif et le serveur de bases de données
* SQL fournit un moyen alternatif d’**exécution de tâches planifiées**: ceci est pratique dans le cas de la mise à jour d’un fichier log permettant de suivre l’exécution des tâches sur la BDs.
* **La maintenance des triggers est moins coûteuse** que celle de l’application. Et, étant donné que le changement des applications est plus récurrent qu’une migration de BD, il devient moins onéreux de mettre à jour des triggers plutôt que de faire des modifications au niveau applicatif
* **Deux applications différentes peuvent profiter de l’intelligence du trigger**: par exemple, pour le cas d’un batch et d’un client lourd, il aurait fallu deux modifications (le code du batch et celui du client) pour respecter la logique métier

**b - Inconvénients de l’utilisation des triggers :**

* **L’exécution des triggers s’effectue de manière transparente à la couche application**, ce qui cause une difficulté à savoir ce qui se passe réellement dans la base de données
* **La portabilité** : l’écriture de triggers rend une application très dépendante de a base de données
* **La dispersion de l’intelligence métier** : en écrivant la logique métier dans le code de l’application et des triggers, on la décrit à deux endroit différents. De plus, les traitements inclus dans les triggers ne sont pas forcément accessibles lors de l’édition du code applicatif, et vice versa. On augmente finalement les possibilités de ne plus s’y retrouver. Il faudra produire et maintenir une documentation de qualité pour faire diminuer ce risque.

Au terme de cette analyse, il ressort que : l’utilisation des triggers sera prépondérante devant la vérification au niveau applicatif quand le nombre le nombre d’applications utilisant la base de données est important, ou lorsque le changement de bases de données s’effectue à un rythme nettement lent comparé à celui du changement d’application métier.

Cependant, il est préférable d’effectuer les tests de validité au niveau applicatif lorsque les contraintes de portabilité de la base de données de l’application primeront pour l’entreprise et aussi afin d’éviter la dispersion de l’intelligence métier.

1. **Création et différents types de triggers**

En ce qui concerne la syntaxe de création et les différents types proposés, ils varient selon le SGBD utilisé. De ce fait, comme nous nous sommes basés dans le « cadre de cet exposé sur les SGBD Oracle et MySQL, nous allons donc rencontrer des divergences entre ces deux SGBD à plusieurs niveaux de l’implémentation. Des spécifications seront néanmoins données afin de permettre une meilleure compréhension.

* Pour créer un trigger dans votre schéma sur **Oracle**, vous devez disposer du privilège CREATE TRIGGER. Pour créer un déclencheur dans un autre schéma, le privilège CREATE ANY TRIGGER est requis. En plus de ces conditions, pour fabriquer un déclencheur d’instances, il faut détenir le privilège ADMINISTER DATABASE TRIGGER.
* Pour créer un déclencheur sur MySQL, vous devez disposer du privilège SUPER.

**a – Syntaxe de Création**

* **Pour Oracle**

D’une manière globale la syntaxe de création est donnée par :

// Description de l’événement traqué

CREATE [OR REPLACE] TRIGGER [schéma.] nomDéclencheur

{BEFORE | AFTER | INSTEAD OF}

{{DELETE | INSERT | UPDATE [OF col1 [,col2]… ] }

[OR {DELETE | INSERT | UPDATE [OF col1 [,col2]… ] }]…

ON {[schema.] nomTable | nomvue}

[REFRENCING

{OLD [AS] nomVieux | NEW [AS] nomNew | PARENT [AS] nomParent }

[OLD [AS] nomVieux | NEW [AS] nomNew | PARENT [AS] nomParent]… ] [FOR EACH ROW]}

|

{événementBase [OR événementBase]… |

actionStructureBase [OR action actionStructureBase]… }

ON {[schema.] SCHEMA | DATABASE}}

// Eventuelle condition

[WHEN (condition)]

// Description de l’acton à réaliser

Bloc PL/SQL (variable BEGIN instructions END ;)

|CALL nomSousProgramme(paramètres)}

Les options de cette commande sont les suivantes :

* BEFORE|AFTER|INSTEAD OF précise la chronologie entre l’action à réaliser par le déclencheur LMD et la réalisation de l’événement.
* DELETE|INSERT|UPDATE précise la nature de l’événement pour les déclencheurs LMD.
* ON([schéma.] nomTable | nom Vue) spécifie la table, ou la vue associée au déclencheur LMD.
* REFERENCING permet de renommer des variables.
* FOR EACH ROW différencie les déclencheurs LMD au niveau ligne ou au niveau état.
* événementBase identifie la nature d’un déclencheur d’instance, d’un déclencheur d’erreur ou d’un déclencheur de connexion.
* actionStructureBase spécifie la nature d’un déclencheur LDD (CREATE, ALTER, DROP etc)
* ON {[schéma.] SCHEMA | DATABASE} précise le champ d’application du déclencheur. DATABASE pour des déclencheurs s’exécutant pour quiconque et SCHEMA pour ceux s’exécutant que pour le schéma courant.
* **Pour Mysql**

D’une manière général la syntaxe de création est donnée par :

// Description de l’événement traqué

CREATE TRIGGER nomDéclencheur

{ BEFORE | AFTER }{ DELETE | INSERT | UPDATE }

ON nomTable

FOR EACH ROW

// Description de l’action à réaliser

{ instruction; |

[etiquette :] BEGIN

instructions;

|END [etiquette] ;}

Les options de cette commande sont les suivantes :

* BEFORE|AFTER précise la chronologie entre l’action à réaliser par le déclencheur LMD et la réalisation de l’événement.
* DELETE|INSERT|UPDATE précise la nature de l’événement pour les déclencheurs LMD.
  + Pour DELETE, le déclencheur examine les événements DELETE et REPLACE
  + Pour INSERT, le déclencheur prend en compte les événements suivants : INSERT, CREATE … SELECT, LOAD DATA et REPLACE.
  + Pour UPDATE, le déclencheur considère seulement l’événement UPDATE.
* ON nomTable spécifie la table, ou la vue associée au déclencheur LMD.
* FOR EACH ROW différencie les déclencheurs LMD au niveau ligne ou au niveau état.
* instruction(s) compose le corps du code déclencheur.

**b - Types de triggers :**

**Déclencheur LMD :** se sont des déclencheurs qui réagissent aux événements de mise à jour particulière de la base (ajout, modification ou suppression dans une table ou dans une vue). On programme ce type de déclencheur lorsqu’on désire exécuter autant de fois le déclencheur qu’il y a de lignes concernées par une mise à jour. Les instructions mises en place dans certains trigger LMD peuvent être exécutées avant l’événement (par la directive BEFORE) ou après l’événement (par la directive AFTER).

Les déclencheurs LMD, plus précisément les déclencheurs lignes, sont les seuls types de déclencheurs sur MySQL. Dans ce cas, ces déclencheurs ont forme du déclencheur ci-dessus.

La syntaxe générale de déclencheur LMD sur Oracle est la suivante :

CREATE [OR REPLACE] TRIGGER [schéma.] nomDéclencheur BEFORE | AFTER | INSTEAD OF {INSERT|UPDATE|DELETE} ON {[schema.] SCHEMA | DATABASE}} {[FOR EACH ROW]}

Bloc PL/SQL (variable BEGIN instructions END ;)

|CALL nomSousProgramme(paramètres)}

* **Déclencheur lignes (row triggers) :** Ce déclencheur est déclaré avec la directive FOR EACH ROW. C’est la seule de forme de trigger disponible sur MySQL où il prend la forme du déclencheur de la section **« syntaxe de création »**. Ce n’est que dans ce type de déclencheur que nous avons accès aux anciennes valeurs et aux nouvelles valeurs des colonnes de la ligne affectée par la mise à jour prévue par l’événement.

Les anciennes valeurs sont repérées par la directive OLD alors que les nouvelles valeurs sont repérées par la directive NEW.

* **Déclencheur d’état (statement trigger) :** Ce déclencheur porte sur l’intégralité de la table sur laquelle il est mis et non pas sur chaque enregistrement particulier de la table. Il est donc pas possible d’avoir accès aux valeurs des lignes mises à jour par l’événement.
* **Déclencheurs INSTEAD OF :** Permet la mise à jour d’une vue multitable qui ne pouvait être modifiée directement par INSERT, UPDATE, ou DELETE. Ce déclencheur programmera des actions au lieu d’insérer, de modifier ou de supprimer une vue. Les déclencheurs INSTEAD OF possèdent les caractéristiques suivantes :
* font intervenir la clause FOR EACH ROW
* ne s’utilisent que sur des vues
* ne font pas intervenir les options BEFORE et AFTER

**Déclencheurs LDD :** se sont des déclencheurs qui sont liés à la modification de la structure de la base et non pas à la modification des données de la base. Voici sa syntaxe :

CREATE [OR REPLACE] TRIGGER [schéma.] nomDéclencheur BEFORE | AFTER {actionStructureBase [OR action actionStructureBase]… } ON { [schema.] SCHEMA | DATABASE } }

Bloc PL/SQL (variable BEGIN instructions END ;)

|CALL nomSousProgramme(paramètres)}

Les principales actions sur la structure de la base prises en compte sont :

* ALTER pour déclencher en cas de modification d’un objet du dictionnaire (table, index, séquence, etc)
* COMMENT pour déclencher en cas d’ajout d’un commentaire.
* DROP pour déclencher en cas de suppresion d’un objet du dictionnaire.
* GRANT pour déclencher en cas d’affectation de privilège à un autre utilisateur ou rôle.
* RENAME pour déclencher en cas de changement de nom d’un objet du dictionnaire.
* REVOKE pour déclencher en cas de révocation de privilège d’un autre utilisateur ou rôle.

**Déclencheur d’instances :** se sont des déclencheurs qui sont liés à des événements spéciaux de la base de données comme :

* Le démarrage ou l’arrêt de la base (*startup* ou *shutdown*)
* Une erreur spécifique (NO\_DATA\_FOUND, DUP\_VAL\_ON\_INDEX, etc),
* Une connexion ou une déconnexion d’un utilisateur etc.

Les évènements précités sont programmés à l’aide des mots-clés STATIC, SHUTDOWN, SUSPEND, SERVEERROR, LOGON, LOGOFF dans la syntaxe suivante :

CREATE [OR REPLACE] TRIGGER [schéma.] nomDéclencheur BEFORE | AFTER {actionStructureBase [OR action actionStructureBase]… } ON { [schema.] SCHEMA | DATABASE}}

Bloc PL/SQL (variable BEGIN instructions END ;)

|CALL nomSousProgramme(paramètres)}

Les restrictions régissant ces déclencheurs sont les suivantes :

* Seule l’option AFTER est valable LOGON, STARTUP, SERVEERROR et SUSPEND.
* Seule l’option BEFORE est valable pour LOGOFF et SHUTDOWN.
* Les options AFTER STARTUP et BEFORE SHUTDOWN s’appliquent seulement sur les déclencheurs de type DATABASE.

1. **Conclusion**
2. **Bibliographie**