Chaque enregistrement qui tente d'être ajouté dans la table `Qualifications` est désigné par `:NEW` au niveau du code du déclencheur. L'accès aux colonnes de ce pseudo-enregistrement dans le corps du déclencheur se fait par la notation pointée.

Le code minimal de ce déclencheur (on ne prend pas en compte l'éventuelle erreur du `SELECT` ne renvoyant aucun pilote) est décrit dans le tableau suivant :

Tableau 7-27 Déclencheur avant insertion

Code PL/SQL	Commentaires

`CREATE TRIGGER TrigInsQualif`
`BEFORE INSERT ON Qualifications`
`FOR EACH ROW`
	Déclaration de l'événement déclencheur.

`DECLARE`
`v_compteur Pilote.nbHVol%TYPE;`
`v_nom Pilote.nom%TYPE;`
	Déclaration des variables locales.

`BEGIN`
`SELECT nbQualif, nom INTO v_compteur, v_nom`
`FROM Pilote WHERE brevet = :NEW.brevet;`
`IF v_compteur < 3 THEN`
`UPDATE Pilote SET nbQualif = nbQualif + 1`
`WHERE brevet = :NEW.brevet;`
`ELSE`
`RAISE_APPLICATION_ERROR(-20100, 'Le pilote '`
`|| v_nom || ' a déjà 3 qualifications!');`
`END IF;`
`END;`
`/`
	Corps du déclencheur. Extraction et mise à jour du pilote concerné par la qualification.
	
	Renvoi d'une erreur utilisateur.

Le test de ce déclencheur peut être réalisé sous SQL*Plus comme le montre la trace suivante. On retrouve l'erreur utilisateur qui est levée en premier.

Tableau 7-28 Test du déclencheur

Événement déclencheur	Sortie SQL*Plus

`SQL> INSERT INTO Qualifications`
`VALUES ('PL-2', 'A380',`
`'20-06-2006');`
	`1 ligne créée.`
	`SQL> SELECT * FROM Pilote;`
	`BREVET NOM NBHVOL COMP NBQUALIF`
	`------ ------------ ------ ---- ---------`
	`PL-1 J.M Misztela 450 AF 3`
	`PL-2 Thierry Guibert 3400 AF 2`
	`PL-3 Michel Tuffery 900 SING 1`

`SQL> INSERT INTO Qualifications`
`VALUES ('PL-1', 'A380',`
`'20-06-2006');`
	`ERREUR à la ligne 1 :`
	`ORA-20100: Le pilote J.M Misztela a déjà 3 qualifications!`
	`ORA-06512: à "SOUTOU.TRIGINSQUALIF", ligne 9`
	`ORA-04088: erreur lors d'exécution du déclencheur 'SOUTOU.TRIGINSQUALIF'`

Comme l'instruction RAISE, la procédure RAISE_APPLICATION_ERROR passe par la section EXCEPTION (s'il en existe une) avant de terminer le déclencheur. En conséquence, si vous utilisez aussi une section exception dans le même bloc, il faut forcer la sortie du déclencheur par la directive RAISE pour ne pas perdre le message d'erreur et surtout ne pas réaliser la mise à jour de la base.

Afin d'illustrer cette importante remarque, ajoutons une section EXCEPTION au précédent exemple. Cette section vérifiera l'existence du pilote.

Tableau 7-29 Déclencheur avec exceptions

Code PL/SQL	Commentaires

`CREATE TRIGGER TrigInsQualif`
`BEFORE INSERT ON Qualifications`
`FOR EACH ROW`
	Déclaration de l'événement déclencheur.

`DECLARE`
`v_compteur Pilote.nbHVol%TYPE;`
`v_nom Pilote.nom%TYPE;`
	Déclaration des variables locales.

`BEGIN`
`SELECT nbQualif, nom INTO v_compteur, v_nom`
`FROM Pilote WHERE brevet = :NEW.brevet;`
`IF v_compteur < 3 THEN`
`UPDATE Pilote SET nbQualif = nbQualif + 1`
`WHERE brevet = :NEW.brevet;`
`ELSE`
`RAISE_APPLICATION_ERROR(-20100, 'Le pilote ' ||`
`:NEW.brevet || ' a déjà 3 qualifications!');`
`END IF;`
	Corps du déclencheur. Extraction et mise à jour du pilote concerné par la qualification.
	
	Renvoi d'une erreur utilisateur.

`EXCEPTION`
`WHEN NO_DATA_FOUND THEN`
`RAISE_APPLICATION_ERROR(-20101, 'Pas de pilote de code brevet ' || :NEW.brevet);`
	Si erreur au SELECT.

`WHEN OTHERS THEN`
`RAISE;`
`END;`
`/`
	Retour de l'erreur courante.

Le test d'erreur de ce déclencheur sous SQL*Plus est illustré dans le tableau suivant :

Tableau 7-30 Test du déclencheur avec exceptions

Événement déclencheur	Sortie SQL*Plus

`SQL> INSERT INTO Qualifications`
`VALUES ('Qui?', 'A380',`
`'20-06-2006');`
	`ERREUR à la ligne 1 :`
	`ORA-20101: Pas de pilote de code brevet Qui?`
	`ORA-06512: à "SOUTOU.TRIGINSQUALIF", ligne 13`
	`ORA-04088: erreur lors d'exécution du déclencheur 'SOUTOU.TRIGINSQUALIF'`

Pour que la cohérence soit plus complète, il faudrait aussi programmer le déclencheur qui décrémente la valeur de la colonne `nbQualif` pour chaque pilote concerné par une suppression de lignes dans la table `Qualifications`. Il faut raisonner ici sur la directive `:OLD`.

Quand utiliser la directive :OLD ?

Chaque enregistrement qui tente d'être supprimé d'une table qui inclut un déclencheur de type `DELETE FOR EACH ROW`, est désigné par `:OLD` au niveau du code du déclencheur. L'accès aux colonnes de ce pseudo-enregistrement dans le corps du déclencheur se fait par la notation pointée.

Programmons le déclencheur `TrigDelQualif` qui surveille les suppressions de la table `Qualifications` et décrémente de 1 la colonne `nbQualif` pour le pilote concerné par la suppression de sa qualification.

L'événement déclencheur est ici `AFTER INSERT` car il faudra s'assurer que la suppression n'est pas entravée par d'éventuelles contraintes référentielles. On utilise un déclencheur `FOR EACH ROW`, car s'il se produit une suppression de toute la table (`DELETE FROM Qualifications;`) on exécutera autant de fois le déclencheur qu'il y a de lignes supprimées.

Le code minimal de ce déclencheur (on ne prend pas en compte le fait qu'il n'existe plus de pilote de ce code brevet) est décrit dans le tableau suivant :

Tableau 7-31 Déclencheur après suppression

Code PL/SQL	Commentaires

`CREATE TRIGGER TrigDelQualif`
`AFTER DELETE ON Qualifications`
`FOR EACH ROW`
	Déclaration de l'événement déclencheur.

`BEGIN`
`  UPDATE Pilote SET nbQualif = nbQualif - 1`
`  WHERE brevet = :OLD.brevet;`
`END;`
`/`
	Corps du déclencheur. Mise à jour du pilote concerné par la suppression.

En considérant les données initiales des tables, le test de ce déclencheur sous SQL*Plus est le suivant :

Tableau 7-32 Test du déclencheur

Événement déclencheur	Sortie SQL*Plus

`SQL> DELETE FROM Qualifications`
`  WHERE typa = 'A320';`
	`2 ligne(s) supprimée(s).`
	`SQL> SELECT * FROM Pilote;`
	`BREVET NOM  NBHVOL COMP  NBQUALIF`
	`------ -------------- ------ ----  -------`
	`PL-1  J.M Misztela  450 AF  2`
	`PL-2  Thierry Guibert  3400 AF  0`
	`PL-3  Michel Tuffery  900 SING  1`

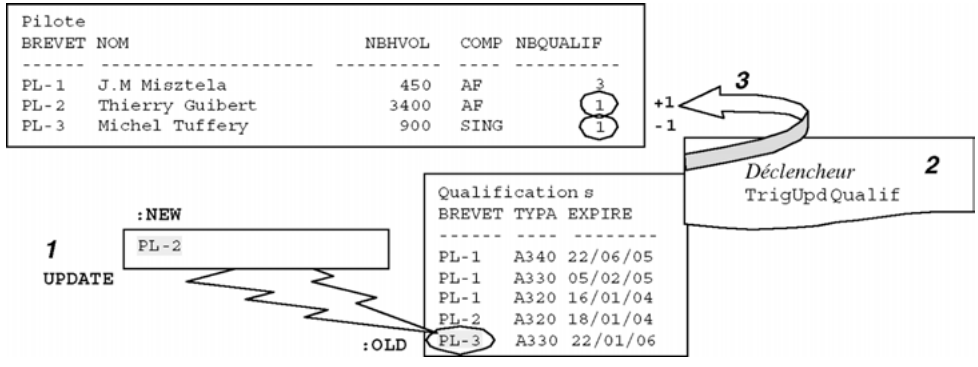
Pour tester le fait que l'instruction UPDATE n'affecte aucune ligne, il faudrait utiliser un curseur implicite (SQL%FOUND) et une erreur utilisateur (voir le paragraphe « Utilisation du curseur implicite » dans la section « Exceptions »).

Quand utiliser à la fois les directives :NEW et :OLD ?

Seuls les déclencheurs de type UPDATE FOR EACH ROW permettent de manipuler à la fois les directives :NEW et :OLD. En effet, la mise à jour d'une ligne dans une table fait intervenir une nouvelle donnée qui en remplace une ancienne. L'accès aux anciennes valeurs se fera par la notation pointée du pseudo-enregistrement :OLD. L'accès aux nouvelles valeurs se fera par :NEW.

La figure suivante illustre ce mécanisme dans le cas de la modification de la colonne brevet du dernier enregistrement de la table Qualifications. Le déclencheur doit programmer deux mises à jour dans la table Pilote.

Figure 7-13 *Principe du déclencheur* TrigUpdQualif



L'événement déclencheur est ici AFTER UPDATE car il faudra s'assurer que la suppression n'est pas entravée par d'éventuelles contraintes référentielles. Le code minimal de ce déclencheur (on ne prend pas en compte le fait qu'un pilote n'ait pas pu être mis à jour) est décrit dans le tableau 7-33.

En considérant les données présentées à la figure précédente, le test de ce déclencheur sous SQL*Plus est présenté dans le tableau 7-34.

Synthèse à propos de :NEW et :OLD

Le tableau 7-35 résume les valeurs contenues dans les pseudo-enregistrements :OLD et :NEW pour les déclencheurs FOR EACH ROW. Retenez que seuls les déclencheurs UPDATE peuvent manipuler à bon escient les deux types de directives.

Attention, Oracle ne vous prévient pas à la compilation que vous utilisez une variable :OLD dans un déclencheur INSERT (ou :NEW dans un déclencheur DELETE), et qui sera toujours nulle.

Tableau 7-33 Déclencheur après modification

Code PL/SQL	Commentaires

`CREATE TRIGGER TrigUpdQualif`
`AFTER UPDATE OF brevet ON Qualifications`
`FOR EACH ROW`
	Déclaration de l'événement déclencheur.

`DECLARE`
`v_compteur Pilote.nbHVol%TYPE;`
`v_nom Pilote.nom%TYPE;`
	Déclaration des variables locales.

`BEGIN`
`SELECT nbQualif, nom INTO v_compteur, v_nom`
`FROM Pilote WHERE brevet = :NEW.brevet;`
	Corps du déclencheur.

`IF v_compteur < 3 THEN`
`UPDATE Pilote SET nbQualif = nbQualif + 1`
`WHERE brevet = :NEW.brevet;`
`UPDATE Pilote SET nbQualif = nbQualif – 1`
`WHERE brevet = :OLD.brevet;`
	Mise à jour des pilotes concernés par la modification de la qualification.

`ELSE`
`RAISE_APPLICATION_ERROR(-20100, 'Le pilote ' ||`
`:NEW.brevet || ' a déjà 3 qualifications!');`
`END IF;`
	Renvoi d'une erreur utilisateur.

`EXCEPTION`
`WHEN NO_DATA_FOUND THEN`
`RAISE_APPLICATION_ERROR(-20101, 'Pas de pilote de code brevet ' || :NEW.brevet);`
	Renvoi d'une erreur utilisateur.

`WHEN OTHERS THEN` `RAISE;`	Retour de l'erreur courante.
`END;` `/`	

Tableau 7-34 Test du déclencheur

Événement déclencheur	Sortie SQL*Plus

`SQL> UPDATE Qualifications`
`SET brevet = 'PL-2'`
`WHERE brevet = 'PL-3'`
`AND typa = 'A330';`
	`1 ligne mise à jour.`
	`SQL> SELECT * FROM Pilote;`
	`BREVET NOM NBHVOL COMP NBQUALIF`
	`------ -------------- ------ ---- --------`
	`PL-1 J.M Misztela 450 AF 3`
	`PL-2` `Thierry Guibert 3400 AF` **`2`**
	`PL-3` `Michel Tuffery 900 SING` **`0`**

Condition dans un déclencheur (WHEN)

Il est possible de restreindre l'exécution d'un déclencheur en amont du code de ce dernier. La clause `WHEN`, placée avant le corps du déclencheur, permet de programmer cette condition. Si celle-ci est réalisée pour l'enregistrement concerné par l'événement, le déclencheur s'exécute. Dans le cas inverse, le déclencheur n'a aucun effet.

Tableau 7-35 Valeurs de :OLD et :NEW

Nature de l'événement	:OLD.*colonne*	:NEW.*colonne*
INSERT	NULL	Nouvelle valeur.
UPDATE	Ancienne valeur.	Nouvelle valeur.
DELETE	Ancienne valeur.	NULL

La condition contenue dans la clause WHEN doit être une expression SQL, et ne peut inclure de requêtes ni de fonctions PL/SQL.

Restreignons par exemple la règle de gestion que nous avons programmée jusqu'à présent – *tout pilote ne peut être qualifié sur plus de trois types d'appareils* – aux appareils de type 'A320', 'A330' ou 'A340'. Il suffira de modifier les en-têtes des trois déclencheurs de la manière suivante (exemple pour le déclencheur d'insertion). Notez que dans la condition WHEN, les « pseudo enregistrements » NEW et OLD s'écrivent sans le symbole « : ».

Tableau 7-36 Déclencheur conditionnel

Code PL/SQL	Commentaires
CREATE OR REPLACE TRIGGER TrigInsQualif BEFORE INSERT ON Qualifications FOR EACH ROW	Déclaration de l'événement déclencheur.
WHEN (NEW.typa = 'A320' OR NEW.typa = 'A340' OR NEW.typa = 'A330')	Condition de déclenchement.

DECLARE
…
BEGIN
…
END;
/
	Corps du déclencheur.

Le tableau suivant présente un jeu de test pour ce déclencheur.

Tableau 7-37 Test du déclencheur

Événement déclencheur	Événement non déclencheur
INSERT INTO Qualifications VALUES ('PL-2',**'A340'**,'20-06-2006');	INSERT INTO Qualifications VALUES ('PL-2',**'A380'**,'20-06-2006');

Corrélation de noms (REFERENCING)

La clause REFERENCING permet de mettre en corrélation les noms des pseudo-enregistrements (:OLD et :NEW) avec des noms de variables. La directive PARENT concerne les déclencheurs

portant sur des collections *nested tables* (extension objet). La condition écrite dans la directive WHEN peut utiliser les noms de variables corrélées.

Utilisons cette clause sur le précédent déclencheur pour renommer le pseudo-enregistrement :NEW par la variable nouveau. Cet enregistrement est opérationnel dans la clause WHEN et dans le corps du déclencheur.

Tableau 7-38 Corrélation de noms

Code PL/SQL	Commentaires

`CREATE OR REPLACE TRIGGER TrigInsQualif`
` BEFORE INSERT ON Qualifications`
` REFERENCING NEW AS nouveau`
` FOR EACH ROW`
` WHEN (nouveau.typa = 'A320' OR`
`  nouveau.typa='A340' OR nouveau.typa='A330')`
`DECLARE`
	Événement déclencheur.
	Renomme :NEW en nouveau.

`DECLARE`
` …`
`BEGIN`
` …`
` …WHERE brevet = :nouveau.brevet;`
`  …`
`END;`
	Corps du déclencheur.

Regroupements d'événements

Des événements (INSERT, UPDATE ou DELETE) peuvent être regroupés au sein d'un même déclencheur s'ils sont de même type (BEFORE ou AFTER). Ainsi, un seul déclencheur est à coder et des instructions dans le corps du déclencheur permettent de retrouver la nature de l'événement déclencheur :

- IF (INSERTING) THEN… exécute un bloc dans le cas d'une insertion ;
- IF (UPDATING('*colonne*')) THEN… exécute un bloc dans le cas de la modification d'une colonne ;
- IF (DELETING) THEN… exécute un bloc en cas d'une suppression.

Utilisons cette fonctionnalité pour regrouper les déclencheurs de type AFTER que nous avons programmés.

Si vous regroupez ainsi plusieurs déclencheurs mono-événements en un déclencheur multi-événements, pensez à supprimer les déclencheurs mono-événements (DROP TRIGGER…) pour ne pas programmer involontairement plusieurs fois la même action par l'intermédiaire des différents déclencheurs existants.

Tableau 7-39 Regroupement d'événements

Code PL/SQL	Commentaires

`CREATE OR REPLACE TRIGGER TrigDelUpdQualif`
`AFTER DELETE OR UPDATE OF brevet ON Qualifications`
`FOR EACH ROW`
	Regroupement de deux événements déclencheurs.

`DECLARE`
`…`
`BEGIN`
`IF (DELETING) THEN`
`…`
	Bloc exécuté en cas de DELETE.

`ELSIF (UPDATING('brevet')) THEN`
`…`
`END IF;`
`END;`
	Bloc exécuté en cas de UPDATE de la colonne brevet.

Déclencheurs d'état (statement triggers)

Un déclencheur d'état est déclaré dans la directive `FOR EACH ROW`. Il n'est pas possible d'avoir accès aux valeurs des lignes mises à jour par l'événement. Le raisonnement de tels déclencheurs porte donc sur la globalité de la table et non sur chaque enregistrement particulier.

Dans le cadre de notre exemple, programmons le déclencheur `périodeOKQualifs` qui interdit toute mise à jour sur la table `Qualifications` pendant les week-ends. Quel que soit le nombre de lignes concernées par un événement, le déclencheur s'exécutera une seule fois avant chaque événement sur la table `Qualifications`.

Tableau 7-40 Déclencheur d'état

Code PL/SQL	Commentaires

`CREATE TRIGGER périodeOKQualifs`
`BEFORE DELETE OR UPDATE`
`OR INSERT ON Qualifications`
	Déclaration des événements déclencheurs.

`BEGIN`
`IF TO_CHAR(SYSDATE,'DAY') IN`
`('SAMEDI', 'DIMANCHE') THEN`
`RAISE_APPLICATION_ERROR(-20102, 'Désolé pas`
`de mises à jour des qualifs le week-end.');`
`END IF ;`
`END;`
`/`
	Bloc exécuté avant chaque mise à jour de la table Qualifications.

Pour chaque actualisation de la table, le déclencheur renvoie le résultat suivant sous SQL*Plus (ça tombe bien, j'ai écrit ce code un dimanche…) :

Tableau 7-41 Test du déclencheur

Événements déclencheurs	Sortie SQL*Plus
UPDATE Qualifications SET …	ERREUR à la ligne 1 : **ORA-20102**: Désolé pas de mises à jour des qualifs le week-end.
INSERT INTO Qualifications VALUES…	ORA-06512: à "SOUTOU.PÉRIODEOKQUALIFS", ligne 3
DELETE FROM Qualifications…	ORA-04088: erreur lors d'exécution du déclencheur 'SOUTOU.PÉRIODEOKQUALIFS'

Déclencheurs INSTEAD OF

Un déclencheur `INSTEAD OF` permet de mettre à jour une vue multitable qui ne pouvait être modifiée directement par `INSERT`, `UPDATE` ou `DELETE` (voir chapitre 5). Nous verrons que seulement certaines vues multitables peuvent être modifiables par l'intermédiaire de ce type de déclencheur. L'expression *instead of* est explicite : le déclencheur programmera des actions *au lieu* d'insérer, de modifier ou de supprimer une vue.

La version 7 d'Oracle n'offrait pas cette possibilité. Ce mécanisme intéresse particulièrement les bases de données réparties par liens (*database links*). Il est désormais plus facile de modifier des informations provenant de différentes tables par ce type de déclencheur.

Caractéristiques

Les déclencheurs `INSTEAD OF` :

- font intervenir la clause `FOR EACH ROW` ;
- ne s'utilisent que sur des vues ;
- ne font pas intervenir les options `BEFORE` et `AFTER`.

L'option de contrôle (`CHECK OPTION`) d'une vue n'est pas vérifiée lors d'un événement (ajout, modification ou suppression) si un déclencheur `INSTEAD OF` est programmé sur cet événement. Le corps du déclencheur doit donc explicitement prendre en compte la contrainte.

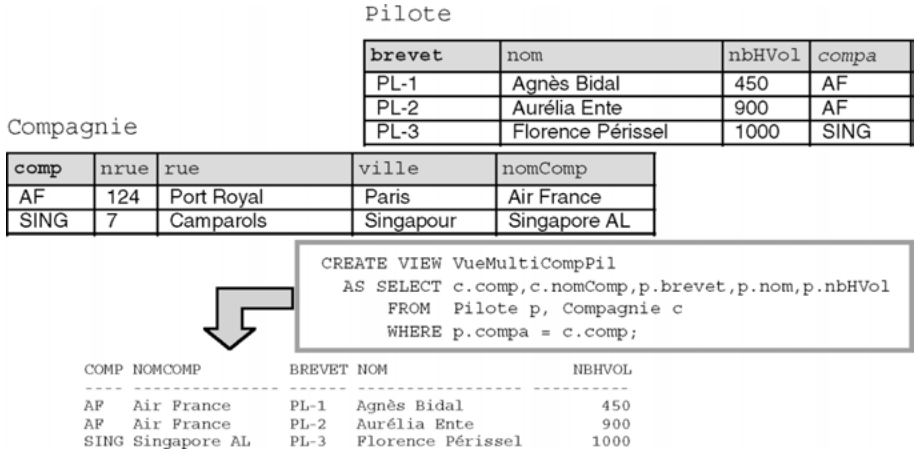
Il n'est pas possible de spécifier une liste de colonnes dans un déclencheur `INSTEAD OF UPDATE`, le déclencheur s'exécutera quelle que soit la colonne modifiée.

Il n'est pas possible d'utiliser la clause `WHEN` dans un déclencheur `INSTEAD OF`.

Exemple

Considérons la vue `VueMultiCompPil` résultant d'une jointure entre les tables `Compagnie` et `Pilote`. Nous avons vu au chapitre 5 que cette vue n'était pas modifiable sous SQL. Nous allons programmer un déclencheur `INSTEAD OF` qui va permettre de la changer de manière transparente.

Figure 7-14 *Vue multitable à modifier*



Le déclencheur qui gère les insertions dans la vue est chargé d'insérer, à chaque nouvel ajout, un enregistrement dans chacune des deux tables.

Tableau 7-42 Déclencheur INSTEAD OF

Code PL/SQL	Commentaires

`CREATE TRIGGER TrigAulieuInsererVue`
`INSTEAD OF INSERT ON VueMultiCompPil`
`FOR EACH ROW`
`DECLARE`
`v_comp NUMBER := 0;`
`v_pil NUMBER := 0;`
	Déclaration de la substitution de l'événement déclencheur.

`BEGIN`
`SELECT COUNT(*) INTO v_pil FROM Pilote`
`WHERE brevet = :NEW.brevet;`
`SELECT COUNT(*) INTO v_comp FROM Compagnie`
`WHERE comp = :NEW.comp;`
`IF v_pil > 0 AND v_comp > 0 THEN`
`RAISE_APPLICATION_ERROR(-20102, 'Le pilote et la compagnie existent déjà!');`
`ELSE`
`IF v_comp = 0 THEN`
`INSERT INTO Compagnie VALUES (:NEW.comp,NULL,NULL,NULL,:NEW.nomComp);`
`END IF;`
`IF v_pil = 0 THEN`
`INSERT INTO Pilote VALUES (:NEW.brevet,:NEW.nom,:NEW.nbHVol,:NEW.comp);`
`END IF;`
`END IF;`
`END;`
`/`
	Corps du déclencheur.
	Cas d'erreur.
	Ajout dans la table `Compagnie`.
	Ajout dans la table `Pilote`.

Pour chaque mise à jour de la vue, le déclencheur insérera un pilote, une compagnie ou les deux, suivant l'existence du pilote et de la compagnie. L'erreur programmée dans le déclencheur concerne le cas pour lequel le pilote et la compagnie existent déjà dans la base. Le tableau suivant décrit une trace de test de ce déclencheur :

Tableau 7-43 Test du déclencheur

Événement déclencheur	Vérification sous SQL*Plus
INSERT INTO VueMultiCompPil VALUES ('AERI', 'Aéris Toulouse', 'PL-4', 'Pascal Larrazet', 5600); 1 ligne créée.	SQL> SELECT * FROM Pilote; BREVET NOM NBHVOL COMP ------ ---------- ---------- ---- PL-1 Agnès Bidal 450 AF … PL-4 Pascal Larrazet 5600 AERI SQL> SELECT * FROM Compagnie; COMP NRUE RUE VILLE NOMCOMP ---- ---- ---------- --------- -------------- - SING 7 Camparols Singapour Singapore AL AF 124 Port Royal Paris Air France AERI Aéris Toulouse

```
SQL> SELECT * FROM VueMultiCompPil;
COMP NOMCOMP          BREVET NOM                  NBHVOL
---- ---------------- ------ -------------------- ----------
AF   Air France       PL-1   Agnès Bidal             450
AF   Air France       PL-2   Aurélia Ente            900
SING Singapore AL     PL-3   Florence Périssel      1000
AERI Aéris Toulouse   PL-4   Pascal Larrazet        5600
```

Transactions autonomes

Un déclencheur peut former une transaction (utilisation possible de `COMMIT`, `ROLLBACK` et `SAVEPOINT`) si la directive `PRAGMA AUTONOMOUS_TRANSACTION` est employée dans la partie déclarative (voir figure 7-1). Une fois démarrée, une telle transaction est autonome et indépendante (voir le début de ce chapitre). Elle ne partage aucun verrou ou ressource, et ne dépend d'aucune transaction principale. Ces déclencheurs autonomes peuvent en outre exécuter des instructions du LDD (`CREATE`, `DROP` ou `ALTER`) en utilisant des fonctions natives de PL/SQL pour le SQL dynamique (voir la section suivante).

Les modifications faites lors d'une transaction autonome deviennent visibles par les autres transactions quand la transaction autonome se termine. Une transaction autonome doit se terminer explicitement par une validation ou une invalidation. Si une exception n'est pas traitée en sortie, la transaction est invalidée.

Déclencheurs LDD

Étudions à présent les déclencheurs gérant les événements liés à la modification de la structure de la base et non plus à la modification des données de la base. Les options BEFORE et AFTER sont disponibles comme le montre la syntaxe générale suivante. La directive DATABASE précise que le déclencheur peut s'exécuter pour quiconque lance l'événement. La directive SCHEMA indique que le déclencheur ne peut s'exécuter que dans le schéma courant.

```
CREATE [OR REPLACE] TRIGGER [schéma.] nomDéclencheur
   BEFORE | AFTER { actionStructureBase [OR actionStructureBase]… }
   ON { [schéma.] SCHEMA | DATABASE } }
   Bloc PL/SQL (variables BEGIN instructions END ; )
   | CALL nomSousProgramme(paramètres) }
```

- Les principales actions sur la structure de la base prise 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.
- CREATE pour déclencher en cas d'ajout d'un objet du dictionnaire.
- DROP pour déclencher en cas de suppression 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.

Le déclencheur suivant interdit toute suppression d'objet, dans le schéma *soutou*, se produisant un lundi ou un vendredi.

Tableau 7-44 Déclencheur LDD

Code PL/SQL	Commentaires
`CREATE TRIGGER surveilleDROPSoutou` `  BEFORE DROP ON soutou.SCHEMA`	Événement déclencheur LDD.

`BEGIN`
`IF TO_CHAR(SYSDATE,'DAY') IN`
`('LUNDI  ','VENDREDI') THEN`
`RAISE_APPLICATION_ERROR(-20104,'Désolé pas de destruction ce jour..') ;`
`END IF ;`
`END;`
`/`
	Corps du déclencheur.
	
	Retour d'une erreur.

Déclencheurs d'instances

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

peuvent être autant d'événements pris en compte par un déclencheur d'instances. Les événements précités sont programmés à l'aide des mots-clés STARTUP, SHUTDOWN, SUSPEND, SERVERERROR, LOGON, LOGOFF, dans la syntaxe suivante :

```
CREATE [OR REPLACE] TRIGGER [schéma.] nomDéclencheur
   BEFORE | AFTER { événementBase [OR événementBase]… }
   ON { [schéma.] SCHEMA | DATABASE } }
   Bloc PL/SQL (variables BEGIN instructions END ; )
   | CALL nomSousProgramme(paramètres) }
```

Les restrictions régissant ces déclencheurs sont les suivantes :

- Seule l'option AFTER est valable pour LOGON, STARTUP, SERVERERROR, 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.

Les erreurs ORA-01403, ORA-01422, ORA-01423, ORA-01034 et ORA-04030 ne sont pas prises en compte par l'événement SERVERERROR.

Le déclencheur suivant insère une ligne dans une table qui indique l'utilisateur et l'heure de déconnexion (sous SQL*Plus, via un programme d'application, etc.). On suppose la table Trace(événement VARCHAR2(100)) créée.

Tableau 7-45 Déclencheurs d'instances

Code PL/SQL	Commentaires
`CREATE TRIGGER espionDéconnexion` `  BEFORE LOGOFF ON DATABASE`	Événement déclencheur.

`BEGIN`
`  INSERT INTO Trace VALUES (USER ||`
`  ' déconnexion le ' ||`
`  TO_CHAR(SYSDATE,'DD-MM-YYYY HH24:MI:SS'));`
`END;`
`/`
	Corps du déclencheur exécuté à chaque déconnexion.

Appels de sous-programmes

Un déclencheur peut appeler directement par CALL (ou dans son corps) un sous-programme PL/SQL ou une procédure externe écrite en C, C++ ou Java. Le tableau suivant décrit quelques appels de sous-programmes qu'il est possible de coder dans un déclencheur (quel que soit son type). On suppose la procédure PL/SQL suivante existante.

```
CREATE PROCEDURE sousProgDéclencheur(param IN VARCHAR2) IS
```

```
BEGIN
   INSERT INTO Trace VALUES ('sousProgDéclencheur (' || param || ')' );
END sousProgDéclencheur;
```

Tableau 7-46 Appels de sous-programmes dans un déclencheur

Code PL/SQL	Commentaires

`CREATE TRIGGER espionConnexion`
` AFTER LOGON ON DATABASE`
` CALL soutou.sousProgDéclencheur(SYSDATE)`
`/`
	Appel direct d'une procédure PL/SQL.

`CREATE TRIGGER TrigDelTrace`
` AFTER SERVERERROR ON soutou.SCHEMA`
`BEGIN`
` sousProgDéclencheur('Une erreur`
`s''est produite');`
`END;`
`/`
	Appel dans le corps du déclencheur d'une procédure PL/SQL.

`CREATE TRIGGER Ex_trig_Java`
` AFTER DELETE ON Compagnie FOR EACH ROW`
`BEGIN`
` DeuxièmeExemple_affiche(:OLD.nomcomp);`
`END;`
`/`
	Appel dans le corps du déclencheur d'un sous-programme Java (voir chapitre 11).

Gestion des déclencheurs

Un déclencheur est actif, comme une contrainte, dès sa création. Il est possible de le désactiver, de le supprimer ou de le réactiver à la demande grâce aux instructions `ALTER TRIGGER` (pour agir sur un déclencheur en particulier) ou `ALTER TABLE` (pour agir sur tous les déclencheurs d'une table en même temps). Le tableau suivant résume les commandes SQL nécessaires à la gestion des déclencheurs :

Tableau 7-47 Gestion des déclencheurs

SQL	Commentaires
ALTER TRIGGER *nomDéclencheur* **COMPILE**;	Recompilation d'un déclencheur.
ALTER TRIGGER *nomDéclencheur* **DISABLE**;	Désactivation d'un déclencheur.
ALTER TABLE *nomTable* **DISABLE ALL TRIGGERS**;	Désactivation de tous les déclencheurs d'une table.
ALTER TRIGGER *nomDéclencheur* **ENABLE**;	Réactivation d'un déclencheur.
ALTER TABLE *nomTable* **ENABLE ALL TRIGGERS**;	Réactivation de tous les déclencheurs d'une table.
DROP TRIGGER *nomDéclencheur*;	Suppression d'un déclencheur.