36.3. Écrire des fonctions déclencheurs en C

Cette section décrit les détails de bas niveau de l'interface d'une fonction déclencheur. Ces informations ne sont nécessaires que lors de l'écriture d'une fonction déclencheur en C. Si vous utilisez un langage de plus haut niveau, ces détails sont gérés pour vous. Dans la plupart des cas, vous devez considérer l'utilisation d'un langage de procédure avant d'écrire vos déclencheurs en C. La documentation de chaque langage de procédures explique comment écrire un déclencheur dans ce langage.

Les fonctions déclencheurs doivent utiliser la « version 1 » de l'interface du gestionnaire de fonctions.

Quand une fonction est appelée par le gestionnaire de déclencheur, elle ne reçoit aucun argument classique, mais un pointeur de « contexte » pointant sur une structure TriggerData. Les fonctions C peuvent vérifier si elles sont appelées par le gestionnaire de déclencheurs ou pas en exécutant la macro :

```
CALLED_AS_TRIGGER(fcinfo)
```

qui se décompose en :

```
((fcinfo)->context != NULL && IsA((fcinfo)->context, TriggerData))
```

Si elle retourne la valeur vraie, alors il est bon de convertir `fcinfo->context` en type `TriggerData *` et de faire usage de la structure pointée TriggerData. La fonction *ne* doit *pas* modifier la structure TriggerData ou une donnée quelconque vers laquelle elle pointe.

struct TriggerData est définie dans `commands/trigger.h` :

```
typedef struct TriggerData
{
    NodeTag         type;
    TriggerEvent    tg_event;
    Relation        tg_relation;
    HeapTuple       tg_trigtuple;
    HeapTuple       tg_newtuple;
    Trigger        *tg_trigger;
    Buffer          tg_trigtuplebuf;
    Buffer          tg_newtuplebuf;
} TriggerData;
```

où les membres sont définis comme suit :

type

Toujours `T_TriggerData`.

tg_event

Décrit l'événement pour lequel la fonction est appelée. Vous pouvez utiliser les macros suivantes pour examiner `tg_event` :

TRIGGER_FIRED_BEFORE(tg_event)

Renvoie vrai si le déclencheur est lancé avant l'opération.

TRIGGER_FIRED_AFTER(tg_event)

Renvoie vrai si le déclencheur est lancé après l'opération.

TRIGGER_FIRED_INSTEAD(tg_event)

Renvoie vrai si le trigger a été lancé à la place de l'opération.

TRIGGER_FIRED_FOR_ROW(tg_event)

Renvoie vrai si le déclencheur est lancé pour un événement en mode ligne.

TRIGGER_FIRED_FOR_STATEMENT(tg_event)

Renvoie vrai si le déclencheur est lancé pour un événement en mode instruction.

TRIGGER_FIRED_BY_INSERT(tg_event)

Retourne vrai si le déclencheur est lancé par une commande **INSERT**.

TRIGGER_FIRED_BY_UPDATE(tg_event)

Retourne vrai si le déclencheur est lancé par une commande **UPDATE**.

TRIGGER_FIRED_BY_DELETE(tg_event)

Retourne vrai si le déclencheur est lancé par une commande **DELETE**.

TRIGGER_FIRED_BY_TRUNCATE(tg_event)

Renvoie true si le trigger a été déclenché par une commande **TRUNCATE**.

tg_relation

Un pointeur vers une structure décrivant la relation pour laquelle le déclencheur est lancé. Voir `utils/rel.h` pour les détails de cette structure. Les choses les plus intéressantes sont `tg_relation->rd_att` (descripteur de nuplets de la relation) et `tg_relation->rd_rel->relname` (nom de la relation ; le type n'est pas char* mais NameData ; utilisez `SPI_getrelname(tg_relation)` pour obtenir un char* si vous avez besoin d'une copie du nom).

tg_trigtuple

Un pointeur vers la ligne pour laquelle le déclencheur a été lancé. Il s'agit de la ligne étant insérée, mise à jour ou effacée. Si ce déclencheur a été lancé pour une commande **INSERT** ou **DELETE**, c'est cette valeur que la fonction doit retourner si vous ne voulez pas remplacer la ligne par une ligne différente (dans le cas d'un **INSERT**) ou sauter l'opération.

tg_newtuple

Un pointeur vers la nouvelle version de la ligne, si le déclencheur a été lancé pour un **UPDATE** et NULL si c'est pour un **INSERT** ou un **DELETE**. C'est ce que la fonction doit retourner si l'événement est un **UPDATE** et que vous ne voulez pas remplacer cette ligne par une ligne différente ou bien sauter l'opération.

tg_trigger

Un pointeur vers une structure de type Trigger, définie dans `utils/rel.h` :

```
typedef struct Trigger
{
    Oid         tgoid;
    char       *tgname;
    Oid         tgfoid;
    int16       tgtype;
    bool        tgenabled;
```

```
    bool        tgisinternal;
    Oid         tgconstrrelid;
    Oid         tgconstrindid;
    Oid         tgconstraint;
    bool        tgdeferrable;
    bool        tginitdeferred;
    int16       tgnargs;
    int16       tgnattr;
    int16      *tgattr;
    char      **tgargs;
    char       *tgqual;
} Trigger;
```

où *tgname* est le nom du déclencheur, *tgnargs* est le nombre d'arguments dans *tgargs* et *tgargs* est un tableau de pointeurs vers les arguments spécifiés dans l'expression contenant la commande **CREATE TRIGGER**. Les autres membres ne sont destinés qu'à un usage interne.

tg_trigtuplebuf

Le tampon contenant *tg_trigtuple* ou InvalidBuffer s'il n'existe pas une telle ligne ou si elle n'est pas stockée dans un tampon du disque.

tg_newtuplebuf

Le tampon contenant *tg_newtuple* ou InvalidBuffer s'il n'existe pas une telle ligne ou si elle n'est pas stockée dans un tampon du disque.

Une fonction déclencheur doit retourner soit un pointeur HeapTuple soit un pointeur NULL (*pas* une valeur SQL NULL, donc ne positionnez pas *isNull* à true). Faites attention de renvoyer soit un *tg_trigtuple* soit un *tg_newtuple*, comme approprié, si vous ne voulez pas changer la ligne en cours de modification.