10. Triggers

As triggers são um recurso muito útil e poderoso do servidor de banco de dados DBMaker. Eles executam automaticamente comandos predefinidos em resposta a eventos específicos, independentemente de qual usuário ou programa de aplicativo os tenha gerado.

As Triggers permitem personalizar um banco de dados de maneiras que podem não ser possíveis com comandos SQL padrão. O banco de dados pode controlar de forma consistente operações complexas ou não convencionais sem exigir ação por parte dos usuários ou programas de aplicativo.

Use as Triggers para:

- Implementar regras de negócios
- Criar um histórico de auditoria para atividades do banco de dados
- Derivar valores adicionais a partir de dados existentes
- Replicar dados em várias tabelas
- Realizar procedimentos de autorização de segurança
- Controlar a integridade dos dados
- Definir restrições de integridade não convencionais

Exerça moderação ao usar triggers para evitar a formação de interdependências complexas no banco de dados que possam ser difíceis de acompanhar e modificar. Utilize triggers apenas quando a funcionalidade desejada não puder ser implementada usando comandos SQL padrão e restrições de integridade.

10.1 Trigger Components

O DBMaker armazena as definições de trigger no catálogo do sistema.

Cada gatilho no DBMaker possui seis componentes principais:

- **Nome da Trigger** — um nome que identifica de forma única a Trigger
- **Tempo de Ação da Trigger** — o momento relativo ao qual a Trigger será acionado
- **Evento da Trigger** — uma situação específica que ocorre no banco de dados em resposta a alguma ação do usuário, como a inserção de dados em uma tabela
- **Tabela da Trigger** — o nome da tabela na qual a Trigger é executada

- **Ação da Trigger**— uma instrução SQL ou um procedimento armazenado que é executado quando o evento a Trigger ocorre
- **Tipo de Trigger** — o tipo de Trigger

Cada um desses componentes deve estar presente em todas as Trigger. Além disso, há um componente opcional, a cláusula **REFERENCING**.

Trigger Name

O nome da Trigger identifica de forma única uma Trigger. Os nomes das Trigger têm um comprimento máximo de 128 caracteres e podem conter letras, números, o caractere de sublinhado e os símbolos # e $. O primeiro caractere não pode ser um número e o nome não pode conter espaços.

Trigger Action Time

O tempo de ação da Triggers especifica se ele deve ser acionado antes ou depois da instrução SQL que o ativa. O tempo de ação da Triggers é especificado pelas palavras-chave **BEFORE** e **AFTER**.

- A palavra-chave **BEFORE** instrui a Triggers a ser acionado antes da instrução da Triggers.
- A palavra-chave **AFTER** instrui a Triggers a ser acionado após a instrução da Triggers.

Apenas um tempo de ação pode ser especificado para cada Triggers.

Trigger Event

O evento dTriggers é a operação do banco de dados que faz com que uma Triggers seja ativado ou acionado. O evento da Triggers pode ser uma instrução **INSERT**, **UPDATE** ou **DELETE** que opera na tabela do gatilho. Apenas um evento de Triggers pode ser especificado para cada instrução de Triggers. No entanto, vários eventos de Triggers podem ser usados para ativar múltiplas Triggers.

Trigger Table

O evento do gatilho opera na tabela de gatilho associada. A tabela de gatilho deve ser uma tabela base; não pode ser uma tabela temporária, visão ou sinônimo. Um gatilho pode ter apenas uma tabela de gatilho.

Trigger Action

A ação do gatilho é o comando que um gatilho executa quando é acionado. A ação do gatilho pode ser uma instrução **INSERT**, **UPDATE**, **DELETE**, **EXECUTE PROCEDURE** ou um bloco de instruções SQL. Um gatilho pode ter apenas uma única ação de gatilho.

Trigger Type

O tipo de Trigger especifica quantas vezes a Trigger será acionado para cada evento de Trigger. Existem dois tipos deTriggers: Trigger de linha e Triggers de instrução. A opção FOR EACH ROW especifica um Triggers de linha, que aciona uma ação de Triggers uma vez para cada linha modificada pelo evento de Trigger. A opção FOR EACH STATEMENT especifica um gatilho de instrução, que aciona uma ação de Triggers uma vez para cada evento de gatilho.

REFERENCING Clause

A cláusula REFERENCING define nomes correlacionados para os valores antigos e novos em uma coluna. Isso é usado principalmente quando os nomes padrão OLD e NEW não podem ser utilizados devido a um conflito de nomes com uma tabela.

10.2 Trigger Operation

O DBMaker verifica se uma Triggers deve ser acionada e executará as Triggers definidas cada vez que um usuário ou um programa de aplicação causar um evento de Triggers. Acionar Triggers dentro de um banco de dados garante que o DBMaker trate os dados de forma consistente em todas as aplicações. Isso garante que, quando um evento específico ocorrer, uma ação relacionada também seja realizada.

Os usuários podem criar Triggers para implementar restrições de integridade de domínio, coluna, referenciais e não convencionais. No entanto, isso também pode ser feito por meio do controle de integridade declarativa. As Triggers não têm um proprietário, mas estão associados a uma tabela.

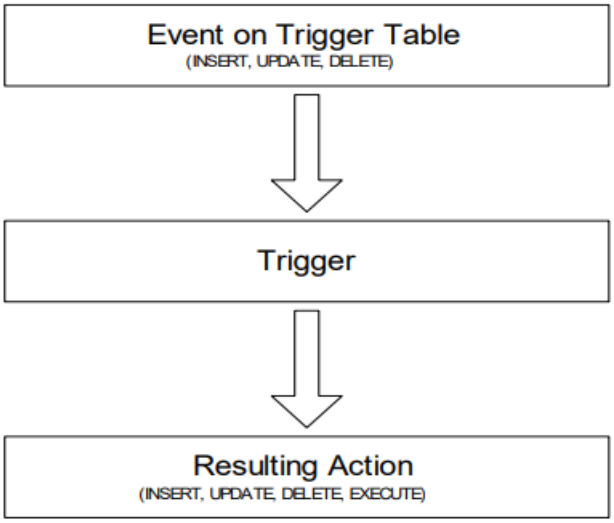


Figure 10-1 Trigger event and action

10.3 Creating Triggers

O comando CREATE TRIGGER cria uma nova Triggers associada a uma tabela específica. Apenas um usuário com privilégio na tabela da Triggers pode executar o comando. O usuário também deve ter os privilégios necessários para todos os objetos referenciados na definição da Triggers para conseguir criar um gatilho com sucesso.

Basic Requirements

Todas as instruções CREATE TRIGGER devem conter pelo menos o seguinte:

- Um nome para a Triggers
- O momento da ação da Triggers (antes ou depois)
- O evento da Triggers
- A tabela da Triggers
- O tipo de Trigger (linha ou instrução)
- A ação da Trigger

Security Privileges

Todas as instruções SQL na ação da Triggers operam com os mesmos privilégios do proprietário da tabela da Triggers, e não com os privilégios do usuário que executa o evento da Triggers. Se o gatilho existir, qualquer usuário que executar o evento da Triggers fará com quea Triggers seja acionada.

CREATE TRIGGER Syntax

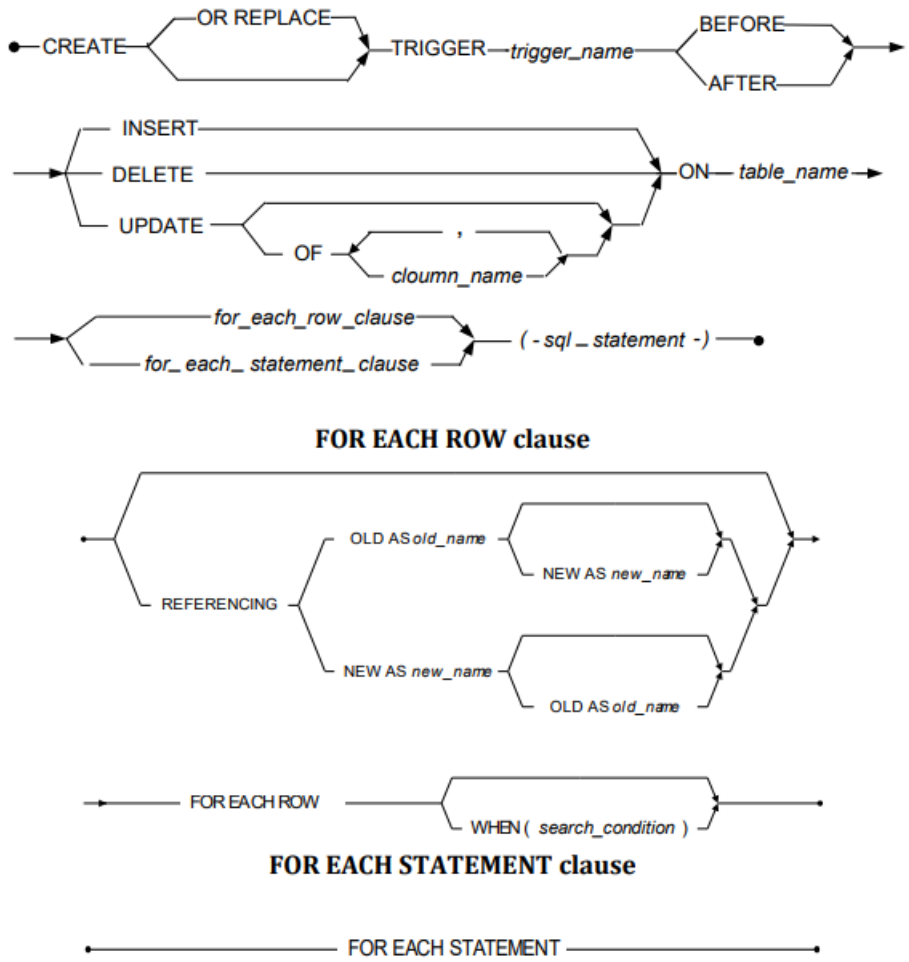


Figure 10-2 Syntax for the CREATE TRIGGER Statement

OR REPLACE é usado para recriar a Trigger que já existe, ou seja, os usuários podem usar essa cláusula para alterar a definição de uma Triggers existente sem excluí-la.

Exemplo:

A seguinte instrução cria ou substitui uma Trigger na tabela **tb_staff:**

```
dmSQL> CREATE OR REPLACE TRIGGER tr_staff_insert AFTER INSERT ON
tb_staff
 FOR       EACH       ROW       WHEN       (new.ID       >       0)
 (INSERT   INTO   tb_salary(new.ID,   new.Name,NULL,   NULL,   NULL));
```

Specifying the Trigger Action Time

Você pode usar a combinação de tempo da Trigger e tipo de Trigger para criar quatro Trigger para cada tabela para o mesmo evento (INSERT, DELETE ou UPDATE). Para cada evento, são possíveis as combinações BEFORE/FOR EACH ROW, AFTER/FOR EACH ROW, BEFORE/FOR EACH STATEMENT e AFTER/FOR EACH STATEMENT.

Uma Trigger BEFORE/FOR EACH STATEMENT executa-se uma vez e somente uma vez antes que a instrução de Trigger seja realizada, ou seja, antes da ocorrência do evento da Trigger. Uma Trigger AFTER/FOR EACH STATEMENT executa-se uma vez e somente uma vez após a conclusão da instrução da Trigger. Note que as Triggers de instrução BEFORE e AFTER são executados mesmo que a instrução das Trigger não processe nenhuma linha.

BEFORE OR AFTER INSERT OR DELETE TRIGGER EVENTS

Os exemplos a seguir mostram como criar Triggers que disparam antes ou depois dos eventos da Trigger INSERT ou DELETE. A ação da Trigger é representada por <sql_statement>.

Exemplo 1:

Para definir quatro Triggers para um evento INSERT na tabela tb:

```
dmSQL> CREATE TRIGGER tr1 BEFORE INSERT ON tb FOR EACH STATEMENT
<sql_statement>;
dmSQL> CREATE TRIGGER tr2 BEFORE INSERT ON tb FOR EACH ROW
<sql_statement>;
dmSQL> CREATE TRIGGER tr3 AFTER INSERT ON tb FOR EACH ROW
<sql_statement>;
dmSQL> CREATE TRIGGER tr4 AFTER INSERT ON tb FOR EACH STATEMENT
<sql_statement>;
```

Exemplo 2:

Para definir quatro Triggers para um evento DELETE na tabela tb:

```
dmSQL> CREATE TRIGGER tr1 BEFORE DELETE ON tb FOR EACH STATEMENT
<sql_statement>;
dmSQL> CREATE TRIGGER tr2 BEFORE DELETE ON tb FOR EACH ROW
<sql_statement>;
dmSQL> CREATE TRIGGER tr3 AFTER DELETE ON tb FOR EACH ROW
<sql_statement>;
```

```
dmSQL> CREATE TRIGGER tr4 AFTER DELETE ON tb FOR EACH STATEMENT
<sql_statement>;
```

BEFORE OR AFTER THE UPDATE TRIGGER EVENT

A situação é diferente para eventos UPDATE. Podem ser criados dois tipos deTriggers UPDATE: Triggers UPDATE <tabela> ou Triggers UPDATE OF <coluna>.

Uma Trigger UPDATE <tabela> é acionado sempre que a tabela é atualizada. Uma Trigger UPDATE OF <coluna> é acionado quando colunas específicas são atualizadas. Pode-se criar uma unica Trigger UPDATE <tabela> ou múltiplos Trigger UPDATE OF <coluna> em uma única tabela. Trigger UPDATE OF <coluna> podem conter várias colunas, mas as colunas em todos as Trigger UPDATE OF <coluna> em uma tabela devem ser mutuamente exclusivas.

Exemplo: Para criar uma Trigger de coluna tr_UpdateColumn na tabela tb_salary nas colunas basepay e bonus, que possui quatro colunas: id, name, basepay e bonus:

```
dmSQL> CREATE TRIGGER Tr_UpdateColumn AFTER UPDATE OF basepay,bonus
ON tb_salary
 FOR EACH ROW
 (INSERT INTO tb_OldSalary VALUES (old.basepay, old.bonus));
```

Se uma segunda Trigger de coluna tr_UpdateBonus que especifica a coluna bonus for criado, o comando falhará porque a coluna bonus já aparece na Trigger tr_UpdateColumn.

```
dmSQL> CREATE TRIGGER tr_UpdateBonus AFTER UPDATE OF bonus,tax ON
tb_salary
 FOR EACH ROW
 (INSERT INTO tb_oldTax VALUES (old.bonus, old.tax));
ERROR (6150): [DBMaker] the insert/update value type is incompatible
with column
data type or compare/operand value is incompatible with column data
type in
expression/predicate
```

Se houver quatro colunas em uma tabela, você pode criar no máximo quatro Trigger de coluna UPDATE ou uma únicoa Trigger de tabela UPDATE, mas não ambos ao mesmo tempo para Trigger do mesmo tipo (por exemplo, BEFORE/FOR EACH ROW).

FOR EACH ROW / FOR EACH STATEMENT Clause

A cláusula FOR EACH STATEMENT especifica que uma Trigger será executada uma vez e somente uma vez para cada evento de Trigger. A Trigger será acionadoa mesmo que a instrução do evento dea Trigger não processe nenhuma linha.

A cláusula FOR EACH ROW especifica que uma Trigger será executada uma vez para cada linha que o evento da Trigger modifica. Se o evento da Trigger não modificar nenhuma linha, a Trigger não será acionada.

As palavras-chave OLD e NEW são usadas para identificar quais valores da tabela da Trigger devem ser usados na ação da Trigger. A palavra-chave OLD indica que os valores da tabela da Trigger antes do evento da Trigger são usados na ação da Trigger. A palavra-chave NEW indica que os valores da tabela da Trigger após o evento de gatilho são usados na ação da Trigger.

Exemplo 1: A seguinte declaração mostra como criar uma Trigger de coluna UPDATE na tabela **tb_Sale**s. O campo totSales é um campo calculado derivado dos dois campos: unitPrice e unitSale. Ambos, unitPrice e unitSale, são colunas que disparam a Trigger.

```
dmSQL> CREATE TRIGGER tr_TotalSale AFTER UPDATE OF unitPrice,
unitSale
ON tb_Sales FOR EACH ROW
 (UPDATE tb_Sales
 SET totSales = new.unitPrice * new.unitSale);
```

Exemplo 2:

No exemplo fornecido, há quatro Trigger.

```
dmSQL> CREATE TRIGGER tr_BeforeUpdatePro BEFORE UPDATE ON tb_Orders
FOR EACH STATEMENT
 (EXECUTE PROCEDURE checkPrivilege);
dmSQL> CREATE TRIGGER tr_BeforeUpdate BEFORE UPDATE ON tb_Orders
 FOR EACH ROW
 (INSERT INTO tb_Old_Value (old.customer, old.amount));
dmSQL> CREATE TRIGGER tr_AfterUpdate AFTER UPDATE ON tb_Orders
 FOR EACH ROW
 (INSERT INTO tb_New_Value (new.customer, new.amount));
dmSQL> CREATE TRIGGER tr_AfterUpdatePro AFTER UPDATE ON Orders
 FOR EACH STATEMENT
```

```
 (EXECUTE PROCEDURE Log_Time);
```

Se um usuário executa uma instrução UPDATE que modifica duas linhas na tabela tb_Orders, o efeito e a ordem de execução são os seguintes:

1. O procedimento checkPrivilege é chamado.
2. Uma linha é inserida na tabela tb_Old_Value.
3. A primeira linha é atualizada.
4. Uma linha é inserida na tabela tb_New_Value.
5. Outra linha é inserida na tabela tb_Old_Value.
6. A segunda linha é atualizada.
7. Outra linha é inserida na tabela tb_New_Value.
8. O procedimento Log_Time é chamado.

Procedimentos armazenados não podem conter instruções de controle de transação como COMMIT, ROLLBACK ou SAVEPOINT. Já as Triggers podem especificar apenas uma única ação acionada, e essa ação deve estar encapsulada entre parênteses.

Using the Referencing Clause

Em Triggers de linha, o <sql_statement> (ou corpo da ação) deve indicar se os valores das colunas usadas são anteriores ou posteriores ao evento da Trigger. Por exemplo, para registrar o preço antigo e o novo preço ao atualizar o preço de um item de venda, use as palavras-chave OLD e NEW, conforme mostrado no exemplo 2 da seção FOR EACH ROW / FOR EACH STATEMENT Clause.

No entanto, em casos raros, as tabelas podem conter colunas com os nomes NEW ou OLD. Se esse for o caso, use a cláusula de referência para definir nomes de correlação. A cláusula de referência permite a criação de dois prefixos que podem ser usados com um nome de coluna: um para referenciar o valor antigo da coluna e outro para referenciar o valor novo. Esses prefixos são chamados de nomes de correlação. Utilize as palavras-chave OLD e NEW para indicar os nomes de correlação.

Exemplo:

```
dmSQL> CREATE TRIGGER tr_log_price AFTER UPDATE OF price ON New
 REFERENCING OLD as pre NEW as post
 FOR EACH ROW
 (INSERT INTO logTbl
 VALUES (item_no, today(), pre.price,
```

```
 post.price));
```

Neste exemplo, o nome da tabela da Trigger é NEW, então os nomes de correlação pre e post são usados no corpo da ação. As cláusulas de referência são válidas apenas para Trigger de linha e não são permitidas em Trigger de instrução.

Se o evento da Trigger for INSERT, não há valor antigo para o registro recém-inserido, portanto, o valor antigo não está disponível. Da mesma forma, se o evento da Trigger for DELETE, não há valor novo para o registro excluído, então o valor novo não está disponível. Para uma Trigger de evento UPDATE, tanto os valores antigos quanto os novos estão disponíveis.

Using the WHEN Condition

Uma cláusula WHEN pode preceder uma ação acionada FOR EACH ROW para tornar a execução da ação dependente do resultado de uma expressão booleana. A cláusula WHEN consiste na palavra-chave WHEN seguida por uma expressão condicional entre parênteses. A cláusula WHEN segue o tempo da ação, mas precede o corpo da ação acionada. A cláusula WHEN não é permitida na definição de uma Trigger de instrução; ela é permitida apenas em Trigger de linha. Exemplo 1:

A seguinte Trigger registra uma reclamação de cliente na tabela tb_logComplain quando um cliente liga para reclamar sobre algo. Suponha que o código da chamada 'c' significa que é uma chamada de reclamação.

```
dmSQL> CREATE TRIGGER tr_log_complain AFTER INSERT ON
tb_Customer_Call
 FOR EACH ROW
 WHEN (new.call_code = 'c')
 (INSERT INTO tb_logComplain
 VALUES (CURRENT_DATE(), Cus_Name));
```

A cláusula WHEN é avaliada para cada linha quando a condição WHEN é incluída na definição da Trigger. Se a condição WHEN avalia para TRUE para uma linha, a ação acionada é executada para essa linha. Se a condição WHEN avalia para FALSE ou desconhecido para uma linha, a ação acionada não é executada para essa linha. O resultado da condição WHEN afeta apenas a execução da ação acionada, não tem efeito na instrução da Trigger.

Exemplo 2: Para criar três Triggers para registrar todas as operações INSERT, UPDATE e DELETE na tabela tb_staff

```
dmSQL> CREATE TRIGGER tr_staff_insert AFTER INSERT ON tb_staff
 FOR EACH ROW
 (INSERT INTO tb_salary
 VALUES (new.Id, new.Name, NULL, NULL, NULL));
dmSQL> CREATE TRIGGER tr_staff_update AFTER UPDATE ON tb_staff
 FOR EACH ROW
 (INSERT INTO tb_staff_bak
 VALUES (old.Id, old.Name,new.Id, new.Name));
dmSQL> CREATE TRIGGER tr_staff_upd AFTER DELETE ON tb_staff
 FOR EACH ROW
 (INSERT INTO tb_staff_bak
 VALUES (old.Id, old.Name,
 NULL, NULL));
```

Exemplo 3: Se a chave primária de uma tabela é alterada, todas as chaves estrangeiras que fazem referência a essa chave primária podem ser atualizadas em cascata. Suponha que deptNo é a chave primária na tabela tb_dept e id é uma chave estrangeira na tabela tb_staff

```
dmSQL> CREATE TRIGGER tr_dept_update BEFORE UPDATE OF deptNo ON
tb_dept
 FOR EACH ROW
 WHEN (NEW.deptNo <> OLD.deptNo)
 (UPDATE tb_staff SET tb_staff.ID = NEW.deptNo
 WHERE tb_staff.ID = OLD.deptNo);
```

Exemplo 4:

Se a chave primária for excluída, todas as chaves estrangeiras podem ser excluídas em cascata.

```
dmSQL> CREATE TRIGGER tr_dept_delete BEFORE DELETE ON tb_dept
 FOR EACH ROW
 (DELETE FROM tb_staff
 WHERE tb_staff.ID = OLD.deptNo);
```

Exemplo 5:

Se uma chave primária for atualizada, todas as chaves estrangeiras podem ser definidas como NULL.

```
dmSQL> CREATE TRIGGER tr_dept_delete BEFORE UPDATE ON tb_dept
 FOR EACH ROW
 (UPDATE tb_staff set ID = NULL
 WHERE tb_staff.ID= OLD.deptNo);
```

Exemplo 6: Se o número de peças em estoque for inferior a um nível determinado, as peças devem ser reordenadas. O número da peça e a quantidade serão registrados em uma tabela chamada tb_pending_orders para ações futuras.

```
dmSQL> CREATE TRIGGER tr_reorder AFTER UPDATE OF parts_on_hand ON
tb_Inventory
 FOR EACH ROW
 WHEN (new.parts_on_hand < new.reorder_level)
 (INSERT INTO tb_pending_orders
 VALUES (new.part_no, new.reorder_qty,
CURRENT_DATE ()));
```

Specifying the Trigger Action

A ação da Trigger é a instrução SQL que é executada quando o evento da Trigger ocorre. A ação da Trigger pode ser uma instrução INSERT, DELETE, UPDATE, EXECUTE PROCEDURE ou um bloco de instruções SQL. Nenhuma outra instrução é permitida.

Procedimentos armazenados não podem conter instruções de controle de transação como COMMIT, ROLLBACK ou SAVEPOINT. Trigger podem especificar apenas uma única ação acionada, que deve estar encapsulada entre parênteses.

Exemplo: A seguinte declaração cria uma Trigger na tabela **tb_staff:**

```
dmSQL> CREATE TRIGGER tr_staff_insert AFTER INSERT ON tb_staff
 FOR EACH ROW WHEN (new.ID > 0)
 (INSERT INTO tb_salary(new.ID, new.Name,NULL,
NULL, NULL));
```

Neste exemplo, o nome do gatilho é tr_staff_insert. A opção AFTER é especificada, o que significa que essa Trigger será acionado após a instrução INSERT ser executada na tabela tb_staff. O evento de gatilho é INSERT, a tabela da Trigger é tb_salary. O tipo de gatilho é FOR EACH ROW. A ação SQL que é acionada é INSERT.

```
dmSQL> CREATE TRIGGER tr_salary_Del AFTER DELETE ON tb_salary
 FOR EACH ROW
 (INSERT INTO tb_old_salry
 VALUES (Old.name));
```

No exemplo acima, a Trigger tr_salry_Del adicionará o nome do cliente excluído na tabela tb_old_salry quando um registro for excluído da tabela tb_salary. Você não pode criar um gatilho em uma tabela temporária, visualização ou tabela do sistema.

10.4 Modifying a Trigger

Uma Trigger não pode ser modificado diretamente, mas sua definição pode ser substituída. Quando você deseja modificar a definição de uma Trigger, use a instrução ALTER TRIGGER.

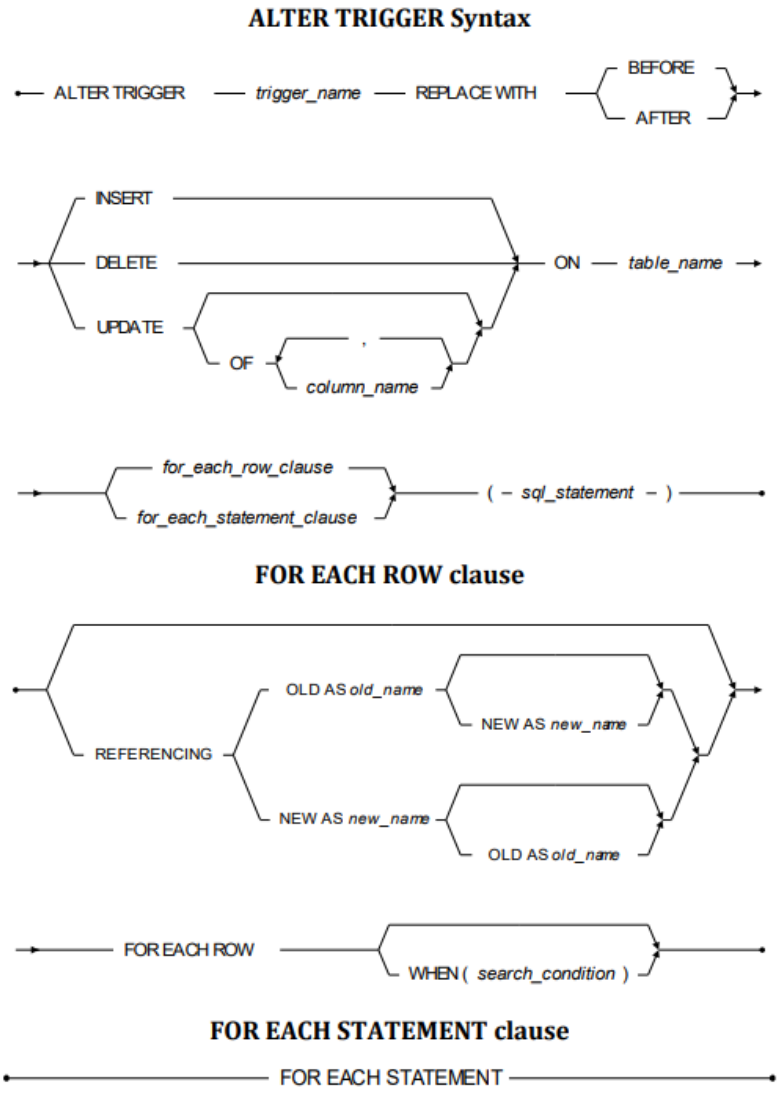


Figure 10-3 Syntax for the ALTER TRIGGER command

Replacing a Trigger Action

Para substituir uma ação de gatilho, use a instrução ALTER TRIGGER tr_name REPLACE WITH...

Exemplo 1:

Se um gerente sair, então os dados dele precisam ser excluídos da tabela tb_manager. Para criar uma Trigger na tabela tb_staff:

```
dmSQL> CREATE TRIGGER tr_staff_del AFTER DELETE ON tb_staff
 FOR EACH ROW
 ( DELETE FROM tb_manager WHERE Id = old.Id );
```

Exemplo 2:

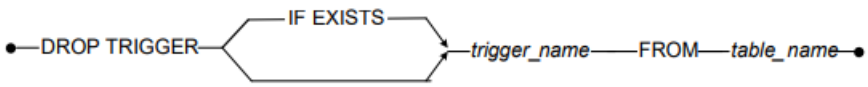
É possível adicionar outra condição à ação da Trigger, como "excluir os dados da tabela tb_manager apenas quando o funcionário for um gerente de projeto." Para substituir uma ação de uma Trigger na tabela tb_staff e adicionar uma condição:

```
dmSQL> ALTER TRIGGER tr_staff_del REPLACE WITH AFTER DELETE ON
tb_staff
 FOR EACH ROW
 ( DELETE FROM tb_manager
 WHERE Id = old.Id
 AND title = 'Project Mananger');
```

Alternativamente, a Trigger pode ser excluída e recriada.

10.5 Dropping a Trigger

A instrução DROP TRIGGER pode ser usada para excluir um gatilho do banco de dados.



Dropping the Trigger

Excluir uma tabela fará com que as Trigger que fazem referência à tabela sejam excluídas. Quando o esquema de uma tabela é alterado, o DBMaker tenta executar a Trigger de acordo com a nova definição da tabela na próxima vez que a Trigger for executada. Se a coluna especificada em um evento ou ação da Trigger for excluída, a execução da Trigger e a instrução falharão. A única solução é excluir a Trigger ou modificar a definição da Trigger de acordo com o novo esquema da tabela. Exclua uma Trigger especificando o nome da Trigger a ser excluído e a tabela associada.

Exemplo 1:

Para excluir a Trigger myTrigger da tabela myTable com o comando DROP TRIGGER:

```
dmSQL> DROP TRIGGER myTrigger FROM myTable;
```

Exemplo 2:

Para excluir a Trigger myTrigger da tabela myTable com o comando DROP TRIGGER IF EXISTS:

```
dmSQL>   DROP   TRIGGER   IF   EXISTS   myTrigger   FROM   myTable;
```

Exemplo 3:

Para criar uma Trigger chamado tr_staff_upd para a tabela tb_staff:

```
dmSQL> CREATE TRIGGER tr_staff_upd AFTER UPDATE ON tb_staff
 FOR EACH ROW
 ( DELETE FROM tb_salary WHERE id = old.id );
```

Se a coluna id na tabela salary for excluída ou o tipo for alterado, ocorrerá um erro de execução quando a instrução da Trigger (atualização na tb_staff) for realizada, fazendo com que o DBMS tente acionar a Trigger tr_staff_upd.

10.6 Using Triggers

Existem várias maneiras de usar as Trigger.

Stored Procedures in Action Body

Uma das funcionalidades mais poderosas das Trigger é a capacidade de usar uma store procedure como uma Trigger. A instrução EXECUTE PROCEDURE chama uma store procedure, permitindo que os dados sejam passados da tabela da Trigger para a store procedure, em seguida, executa o procedimento.

Exemplo:

Para criar uma Trigger e usar a instrução EXECUTE PROCEDURE:

```
dmSQL> CREATE TRIGGER tr_sales_update AFTER UPDATE OF price ON
tb_Sales
 FOR EACH ROW
 (EXECUTE PROCEDURE
 logPrice(item_no, new.price, old.price));
```

Os usuários podem passar valores para um procedimento armazenado na lista de argumentos. Se a chamada das store procedure for parte da ação para uam Trigger

de linha, os usuários podem usar os valores de correlação OLD e NEW para passar os valores das colunas. Se as store procedure for parte de uma Trigger de instrução, apenas constantes podem ser passadas para a store procedure.

Dentro de uma ação da Trigger, você pode atualizar colunas que não acionam a Trigger na tabela de Trigger, com ou sem uma store procedure. Umm store procedure acionada por uma Trigger não pode conter instruções de controle de transação, como BEGIN WORK, COMMIT WORK, ROLLBACK WORK, SAVEPOINT ou instruções DDL.

A store procedure como ação de trigger não pode ser uma procedure superficial que retorne mais de uma linha.

SQL Block in Action Body

Os triggers também podem usar um bloco SQL de trigger como ação do trigger. O bloco SQL de trigger contém um conjunto de instruções SQL que podem ser temporariamente criadas e executadas por um banco de dados.

A ação e a sintaxe de um bloco SQL de trigger são semelhantes às de um bloco SQL anônimo. Um bloco SQL de trigger permite que os usuários executem um conjunto de instruções SQL (um lote de SQL) quando a ação do trigger é executada, e suporta todos os blocos de sintaxe SQL, incluindo variáveis, lógica de gramática e cursores, etc. Ao mesmo tempo, as instruções SQL no bloco SQL ainda podem usar a cláusula OLD \ NEW REFERENCING.

Se os usuários adicionarem um valor OLD \ NEW ao bloco de trigger, a ação do trigger na coluna se referirá a todas as colunas, incluindo os dados que os usuários não inseriram, atualizaram ou excluíram. Quando os usuários alteram o valor da cláusula OLD \ NEW REFERENCING no bloco SQL do trigger, o comportamento é executado automaticamente, uma vez que os eventos do trigger ocorreram.

Existem algumas limitações no bloco SQL de trigger:

- Não há coluna antiga antes/depois da ação de trigger INSERT.
- Não há nova coluna antes/depois da ação de trigger DELETE.
- Não suporta coluna JSONCOLS.
- Verificação de diferentes sintaxes de bloco SQL.

Um bloco SQL de trigger contém mais de uma instrução SQL, e cada instrução termina com ';'. Portanto, o dmSQL deve suportar delimitadores de bloco. O delimitador de bloco pode ser uma série de a-z, A-Z, @, %, ^, e deve conter no mínimo dois caracteres e no máximo sete caracteres. No delimitador de bloco, ';' não denota o fim da entrada. Os usuários devem definir o delimitador de bloco antes de escrever

o bloco SQL de trigger no dmSQL. Caso contrário, um erro será retornado. Para mais informações sobre variáveis e lógica de sintaxe de blocos SQL, consulte o Capítulo 12, Anonymous Stored Procedures

NOTA: *As instruções compostas de um bloco SQL de trigger são delimitadas pelas palavras-chave "BEGIN" e "END" com o delimitador de bloco*.

Exemplo 1:

Para criar um bloco SQL em uma ação de trigger:

```
dmSQL> SET BLOCK DELIMITER @@;
dmSQL> CREATE TABLE tab_t1(c1 int,c2 int);
dmSQL> CREATE TABLE tab_t2(c1 int,c2 int);
dmSQL> INSERT INTO tab_t1 values(111,222);
dmSQL> @@
CREATE TRIGGER uptrg BEFORE UPDATE ON tab_t1 FOR EACH ROW
BEGIN
INSERT INTO tab_t2 values(new.c1,new.c2);
INSERT INTO tab_t2 values(old.c1,old.c2);
END;
@@
dmSQL> UPDATE tab_t1 SET c1= 1 WHERE c1 = 111;
dmSQL> SELECT * FROM tab_t1;
dmSQL> SELECT * FROM tab_t2;
```

Exemplo 2:

Se uma tabela contiver colunas chamadas NEW ou OLD, o bloco SQL da trigger suporta a cláusula REFERENCING.

```
dmSQL> CREATE TABLE tk_tb1(c1 int,new int,old int);
dmSQL> CREATE TABLE tk_tb2(c1 int,new int,old int);
dmSQL> SET BLOCK DELIMITER @@;
dmSQL> @@
CREATE TRIGGER uptrg BEFORE UPDATE ON tk_tb1
REFERENCING OLD as pre NEW as post
FOR EACH ROW
BEGIN
INSERT INTO tk_tb2 values(pre.c1, pre.new, pre.old);
INSERT INTO tk_tb2 values(post.c1, post.new, post.old);
```

Exemplo 3: Editando a cláusula de referência NEW, c1, c2 e c3 são os nomes das colunas antes de inserir os dados na tabela test1:

```
inserting the data to table test1:
dmSQL> CREATE TABLE test1(c1 int,c2 varchar(200),c3 double);
dmSQL> SET BLOCK DELIMITER @@;
dmSQL> @@
CREATE OR REPLACE TRIGGER tg1 BEFORE INSERT ON test1 FOR EACH ROW
BEGIN
SET NEW.c1 = NEW.c1+NEW.c1;
SET NEW.c2 = NEW.c2||NEW.c2;
SET NEW.c3 = NEW.c3+NEW.c3;
END;
@@
dmSQL> INSERT INTO test1 values(1,'old data',100.001);
1 rows inserted
dmSQL> INSERT INTO test1 values(2,'old part1',1.1);
1 rows inserted
dmSQL> INSERT INTO test1 values(3,'oldrow',99.99);
1 rows inserted
dmSQL> SELECT * FROM test1;
C1 C2 C3
===== ================================= ====================
 2 old dataold data 2.0e+002
 4 old part1old part1 2.2e+000
 6 oldrowoldrow 2.0e+002
3 rows selected
```

Editando a cláusula de referência NEW, c1, c2 e c3 são os nomes das colunas após a inserção dos dados na tabela test1:

```
dmSQL> CREATE TABLE test1(c1 int,c2 varchar(200),c3 double);
dmSQL> CREATE TABLE test2(c1 int,c2 varchar(200),c3 double);
dmSQL>@@
CREATE OR REPLACE TRIGGER tg2 AFTER INSERT ON test2 for each row
BEGIN
SET NEW.c1 = NEW.c1+NEW.c1;
SET NEW.c2 = NEW.c2||NEW.c2;
SET NEW.c3 = NEW.c3+NEW.c3;
INSERT INTO test1 values(new.c1, new.c2, new.c3);
END;
@@
```

```
dmSQL> INSERT INTO test2 values(4,'old data',100.001);
1 rows inserted
dmSQL> SELECT * FROM test2;
C1 C2 C3
===== ========================== ==============
 4 old data 1.0e+002
1 rows selected
dmSQL> SELECT * FROM test1;
C1 C2 C3
===== ============================ ================
 8 old dataold data 2.0e+002
1 rows selected
```

Editando a cláusula de referência NEW e OLD, c1, c2 e c3 são os nomes das colunas antes de atualizar os dados na tabela test1:

```
dmSQL> CREATE TABLE test2(c1 int,c2 varchar(200),c3 double);
dmSQL>@@
CREATE OR REPLACE TRIGGER tg2 AFTER INSERT ON test2 for each row
BEGIN
SET NEW.c1 = NEW.c1+NEW.c1;
SET NEW.c2 = NEW.c2||NEW.c2;
SET NEW.c3 = NEW.c3+NEW.c3;
INSERT INTO test1 values(new.c1, new.c2, new.c3);
END;
@@
dmSQL> INSERT INTO test2 values(4,'old data',100.001);
1 rows inserted
dmSQL> SELECT * FROM test2;
C1 C2 C3
===== ========================== ==============
 4 old data 1.0e+002
1 rows selected
dmSQL> SELECT * FROM test1;
C1 C2 C3
===== ============================ ================
 8 old dataold data 2.0e+002
1 rows selected
```

Editando a cláusula de referência NEW e OLD, c1, c2 e c3 são os nomes das colunas antes de atualizar os dados na tabela test1:

```
dmSQL> CREATE TABLE test1(c1 int,c2 varchar(200),c3 double);
dmSQL> CREATE TABLE test2(c1 int,c2 varchar(200),c3 double);
dmSQL> SET BLOCK DELIMITER @@;
dmSQL> @@
CREATE TRIGGER tg1 BEFORE UPDATE ON test1 FRO EACH ROW
BEGIN
SET NEW.c1 = NEW.c1+NEW.c1;
SET OLD.c1 = OLD.c1+OLD.c1;
SET NEW.c2 = NEW.c2||NEW.c2;
SET OLD.c2 = OLD.c2||OLD.c2;
SET NEW.c3 = NEW.c3+NEW.c3;
SET OLD.c3 = OLD.c3+OLD.c3;
INSERT INTO test1 values(new.c1, new.c2, new.c3);
INSERT INTO test2 values(old.c1, old.c2, old.c3);
 END;
 @@
dmSQL> INSERT INTO test1 values(99,'old data',100.001);
1 rows inserted
dmSQL> SELECT * FROM test1;
C1 C2 C3
===== ========================== ==============
 99 old data 1.0e+002
1 rows selected
dmSQL> UPDATE test1 SET c1 = 300,c2 ='new regre2',c3 =199.9991;
1 rows updated
dmSQL> SELECT * FROM test1;
C1 C2 C3
===== ============================ ==============
 600 new regre2new regre2 4.0e+002
 600 new regre2new regre2 4.0e+002
2 rows selected
dmSQL> SELECT * FROM test2;
C1 C2 C3
===== ======================== ================
 198 old dataold data 2.0e+002
1 rows selected
```

Editando a cláusula de referência NEW e OLD, c1, c2 e c3 são os nomes das colunas após atualizar os dados na tabela test1:

```
dmSQL> CREATE TABLE test1(c1 int,c2 varchar(200),c3 double);
dmSQL> CREATE TABLE test2(c1 int,c2 varchar(200),c3 double);
```

```
dmSQL> SET BLOCK DELIMITER @@;
dmSQL> @@
CREATE TRIGGER tg2 AFTER UPDATE ON test1 FRO EACH ROW
BEGIN
SET NEW.c1 = NEW.c1+NEW.c1;
SET OLD.c1 = OLD.c1+OLD.c1;
SET NEW.c2 = NEW.c2||NEW.c2;
SET OLD.c2 = OLD.c2||OLD.c2;
SET NEW.c3 = NEW.c3+NEW.c3;
SET OLD.c3 = OLD.c3+OLD.c3;
INSERT INTO test1 values(new.c1, new.c2, new.c3);
INSERT INTO test2 values(old.c1, old.c2, old.c3);
END;
@@
dmSQL> INSERT INTO test1 values(99,'old data',100.001);
1 rows inserted
dmSQL> SELECT * FROM test1;
C1 C2 C3
===== ======================== ================
 99 old data 1.0e+002
1 rows selected
dmSQL> UPDATE test1 SET c1 = 300,c2 ='new regre2',c3 =199.9991;
1 rows updated
dmSQL> SELECT * FROM test1;
C1 C2 C3
===== ======================== ================
 99 old data 1.0e+002
1 rows selected
dmSQL> UPDATE test1 SET c1 = 300,c2 ='new regre2',c3 =199.9991;
1 rows updated
dmSQL> SELECT * FROM test1;
C1 C2 C3
===== ======================== ================
 300 new regre2 2.0e+002
 600 new regre2new regre2 4.0e+002
2 rows selected
dmSQL> SELECT * FROM test2;
C1 C2 C3
===== ======================== ================
 198 old dataold data 2.0e+002
1 rows selected
```

Editando a cláusula de referência OLD, c1, c2 e c3 são os nomes das colunas antes de excluir os dados na tabela test1:

```
dmSQL> CREATE TABLE test1(c1 int,c2 varchar(200),c3 double);
dmSQL> SET BLOCK DELIMITER @@;
dmSQL> @@
CREATE TRIGGER tg1 BEFORE DELECT ON test1 FOR EACH ROW
BEGIN
SET OLD.c1 = OLD.c1+10000;
SET OLD.c2 = OLD.c2||OLD.c2;
SET OLD.c3 = OLD.c3+OLD.c3;
INSERT INTO test1 values(old.c1, old.c2, old.c3);
END;
@@
dmSQL> INSERT INTO test1 values(1,'old data',100.001);
1 rows inserted
dmSQL> SELECT * FROM test1;
C1 C2 C3
===== ======================== ================
 1 old data 1.0e+002
1 rows selected
dmSQL> DELECT FROM test1 WHERE c1 < 10000;
1 rows deleted
dmSQL> SELECT * FROM test1;
C1 C2 C3
===== ======================== ================
10001 old dataold data 2.0e+002
1 rows selected
```

Editando a cláusula de referência OLD, c1, c2 e c3 são os nomes das colunas após a exclusão dos dados na tabela test1:

```
dmSQL> DROP TABLE IF EXISTS test1;
dmSQL> CREATE TABLE test1(c1 int,c2 varchar(200),c3 double);
dmSQL> @@
CREATE TRIGGER tg2 AFTER DELETE ON test1 FOR EACH ROW
BEGIN
SET OLD.c1 = OLD.c1+OLD.c1;
SET OLD.c2 = OLD.c2||OLD.c2;
SET OLD.c3 = OLD.c3+OLD.c3;
INSERT INTO test1 values(old.c1, old.c2, old.c3);
END;
```

```
@@
dmSQL> INSERT INTO test1 values(2,'old data',100.001);
1 rows inserted
dmSQL> SELECT * FROM test1;
C1 C2 C3
===== ======================== ===============
 2 old data 1.0e+002
1 rows selected
dmSQL> DELETE FROM test1 WHERE c1 < 10000;
1 rows deleted
dmSQL> SELECT * FROM test1;
C1 C2 C3
===== ======================== ===============
 4 old dataold data 2.0e+002
1 rows selected
```

Trigger Execution Order

Os números das colunas de acionamento determinam a ordem de execução das triggers. A execução da trigger começa com o trigger que tem o menor número de coluna de acionamento e prossegue em ordem até o número mais alto. No exemplo a seguir, a = coluna1, b = coluna2, c = coluna3 e d = coluna4.

Exemplo: A operação UPDATE t1 SET b = b + 1, c = c + 1 acionará ambos as triggers. A Trigger trig1, que tem um número de coluna de acionamento menor do que o trig2, será executado primeiro. O exemplo considera quatro colunas chamadas a, b, c e d na tabela t1.

```
dmSQL> CREATE TRIGGER trig1 AFTER UPDATE OF a,c ON t1
 FOR EACH STATEMENT (UPDATE t2 set c1=c1+1);
dmSQL> CREATE TRIGGER trig2 AFTER UPDATE OF b,d ON t1
 FOR EACH STATEMENT (UPDATE t2 set c2=c2+1);
```

Security and Triggers

Primeiramente, o usuário deve ter permissão para executar o evento da trigger; caso contrário, o usuário não conseguirá acionar o evento. No entanto, o usuário não precisa ter permissão para executar a ação desencadeada, pois as instruções SQL na ação desencadeada operam sob o privilégio de domínio do proprietário da Trigger. Uma vez que uma Trigger é criada com sucesso, o criador da Trigger tem o privilégio

de executar a ação desencadeada. Qualquer outra pessoa que possa emitir a instrução de acionamento também pode acionar a Trigger.

Exemplo:

O usuário B pode atualizar ambas as tabelas T1 e T2, e o usuário A pode atualizar T1, mas não T2. Agora, o usuário B cria um trigger para atualização de T1, e a ação desencadeada atualiza T2. Quando o usuário A atualiza T1, a ação desencadeada (atualização de T2) é executada com sucesso, pois a ação desencadeada está sendo executada sob o privilégio de domínio do usuário B. Esta regra de segurança simplifica a execução e elimina a necessidade de o usuário ter mais privilégios para executar a ação desencadeada.

Cursors and Triggers

As instruções UPDATE ou DELETE dentro de um cursor agem de maneira diferente em comparação com uma única instrução de atualização ou exclusão. A Trigger será executada completamente para cada atualização ou exclusão com a cláusula WHERE CURRENT OF.

Por exemplo, se quatro linhas forem alteradas com um cursor, as Triggers BEFORE/FOR EACH STATEMENT, BEFORE/FOR EACH ROW, AFTER/FOR EACH STATEMENT e AFTER/FOR EACH ROW serão executados uma vez para cada linha, totalizando quatro execuções.

Cascading Triggers

Executar uma Trigger pode causar a execução de outra Trigger. Você pode usar Triggers em cascata para impor a integridade referencial. **O DBMaker suporta um máximo de 64 Triggers em cascata**

Exemplo: Para primeiro excluir um cliente da tabela tb_customer, acione a ação para excluir os registros relacionados ao cliente na tabela tb_order, o que, por sua vez, acionará a ação para excluir os registros relacionados ao pedido na tabela tb_item:

```
dmSQL> CREATE TRIGGER tr_cas1 AFTER DELETE ON tb_customer
 FOR EACH ROW
 (DELETE FROM tb_orders WHERE cust_num = old.cust_num);
```

No DBMaker, se os usuários criarem Triggers recursivas, não será retornado um erro no momento da criação do Trigger. No entanto, os usuários receberão um erro quando

os Triggers recursivos forem executados e atingirem o limite máximo de níveis de trigger em cascata.

10.7 Enabling and Disabling Triggers

Quando uma Trigger é criada, ela está no modo ativada, o que significa que a ação acionada é executada quando o evento de disparo ocorre. Às vezes, os usuários podem precisar desativar uma Trigger:

- Quando precisam carregar uma grande quantidade de dados, desativar temporariamente as Trigger acelerará a operação de carregamento.
- Quando os objetos referenciados em uma Trigger estão indisponíveis.

Exemplo 1:

Para desativar a Trigger "Mytrigger" da tabela "Mytable":

```
dmSQL> ALTER TRIGGER Mytrigger ON Mytable DISABLE;
```

Exemplo 2:

Para ativar a Trigger "Mytrigger" na tabela "Mytable":

```
dmSQL> ALTER TRIGGER mytrigger ON mytable ENABLE;
```

Em resumo, uma Trigger possui dois modos possíveis:

- Ativado — A Trigger é ativada ao ser criada, e a ação acionada é executada quando o evento ocorre.
- Desativado — A Trigger desativada não é executado, mesmo que o evento ocorra.

10.8 Create Trigger Privileges

Para criar uma Trigger para uma tabela, o usuário deve ser o proprietário da tabela ou o DBA. O criador da Trigger deve ter privilégios em todos os objetos referenciados na instrução CREATE TRIGGER para que a criação seja bem-sucedida.

No DBMaker, uma Trigger não possui um proprietário; ela é associada a uma tabela. O proprietário da tabela e o DBA têm todos os privilégios relacionados a uma Trigger. Eles podem criar, excluir ou alterar os Trigger.

As instruções SQL na ação da Trigger operam sob os privilégios de domínio do proprietário da Trigger, e não sob os privilégios de domínio do usuário que executa o evento da Trigger.