An Introduction to SQL Triggers

One of the most important tools in SQL are SQL Triggers, which are a special type of stored procedure that can be associated with a table and a SQL data modification operation such as Insert, Update, or Delete. Triggers can be used to cause some action to occur whenever a data change takes place on a table. Additionally, a Trigger can be used to cause cascading changes from one table to another to enforce referential integrity, which is a key concept in database design.

In this article, we'll discuss SQL Triggers in detail to help understand what they are and how they work.

Referential Integrity

An important concept for SQL Triggers is referential integrity

Put simply, referential integrity is a restriction or constraint in the database that enforces the relationship between two tables. The referential integrity constraint requires that values in a foreign key column must either be present in the primary key that is referenced by the foreign key or they must be null.

Database servers can ensure referential integrity defined through Primary Key and Foreign Key constraints is not violated. Inherently, there are no provisions in most database server implementations for cascading key changes and deletion of child rows, when a parent row is removed from a table. Instead, triggers can be defined to fill this void.

Primary and Foreign Key

To understand referential integrity, it is important to understand the distinction between a primary and foreign Key.

A Primary Key is a relational database table column (or combination of columns) designated to uniquely identify each table record. Primary keys must contain unique values, and cannot contain NULL values. A table cannot have more than one Primary Key.

A Foreign Key is the column of a table (or combination of columns) whose values must match values of a column (or combination of columns) of a Primary Key in some other table. They effectively act as a cross-reference between the tables.

Primary/Foreign Key Example

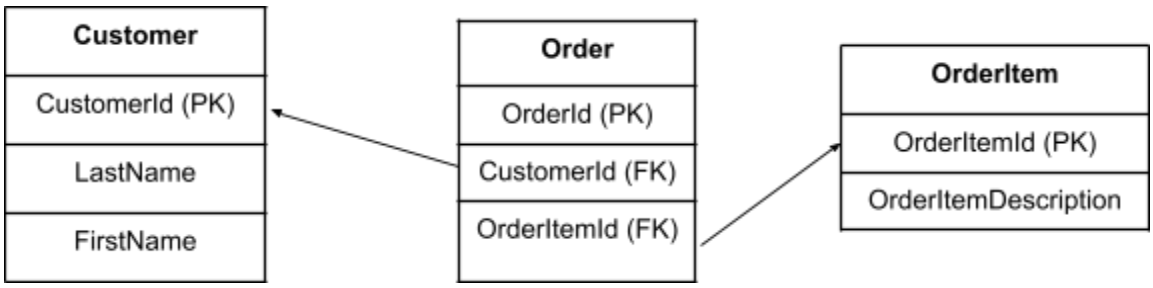
To better understand this concept, consider a 3-table database that consists of Customer, Order, and OrderItem tables. An SQL Server example of a Primary Key and Foreign Key definitions are as follows:

```
CREATE TABLE Customer (
        CustomerId    INT     NOT NULL PRIMARY KEY;
        LastName      VARCHAR(50)       NOT NULL,
        FirstName     VARCHAR(30)       NOT NULL);


CREATE TABLE Order (
        OrderId INT NOT NULL PRIMARY KEY,
        CustomerId INT FOREIGN KEY REFERENCES Customer(CustomerId),
        OrderItemId NOT NULL FOREIGN KEY REFERENCES
                                    OrderItem(OrderItemId);

CREATE TABLE OrderItem (
        OrderItemId           INT     NOT NULL PRIMARY KEY,
        OrderItemDescription VARCHAR(255));
```

Pictorially, the key associations between the 3 tables can be described below:



Let us unpack the definitions of each of the 3 tables above, where “(PK)” designates Primary Keys, and “(FK)” designates Foreign Keys.

First of all, CustomerId is a unique value associated with each customer (row) in Table Customer. Hence, CustomerId is defined as a Primary Key for Table Customer.

Next, OrderItemId is a unique value associated with each order item (row) in Table OrderItem. Hence, OrderItemId is a Primary Key for Table OrderItem.

Next, OrderId is a unique value associated with each order (row) in Table Order. Hence, OrderId is a Primary Key for Table Order. CustomerId in Table Order is designated as a Foreign Key, since its value directly maps to the Primary Key CustomerId in Table Customer. OrderItemId in Table Order is also designated as a Foreigh Key, since its value directly maps to the Primary Key in Table OrderItem.

NOTE: The syntax for defining Primary Keys and Foreign Keys is slightly different in MySQL, but

Identical in concept. Please consult a MySQL reference manual for the subtle syntactical differences.

What are Triggers?

In order to preserve the integrity of a database in terms of maintaining table relationships that have (key) associations, we utilize the concept of Triggers. A database Trigger takes the form of procedural code that is automatically executed in response to certain events on a particular table or view in a database. Again, the Trigger is almost exclusively used for maintaining the integrity of the information in the database.

Triggers can be defined to run Instead Of or After DML (Data Manipulation Language) actions such as Insert, Update, and Delete statements.

For example, in the 3-table model above, if a particular Customer row (referenced by CustomerId) is deleted from Table Customer, any reference (row) to that CustomerId in Table Order would also have to be deleted as well, to maintain the integrity of the database. Failure to remove the associated record (row) in Table Order would violate the integrity of the database, since a CustomerId would be referenced in Table Order, without having a corresponding record in Table Customer (that is, a dangling record).

Types of Triggers

There are 2 types of Triggers that are implemented in most commercial relational database systems:

Instead Of Triggers (also known as Before Triggers):

Instead Of (or Before Triggers), as their name implies, run in place of the DML action which caused them to fire off initially (typically, Insert, Update, or Delete statements). Things to be cognizant of when using Instead Of Triggers include:

1) An Instead Of Trigger always overrides the triggering action. If an Instead Of Trigger trigger is defined to execute on an INSERT statement, then once the INSERT statement attempts to execute, control is immediately passed to the Instead Of Trigger, thus effectively overriding the Insert statement;
2) At most, one Instead Of Trigger can be defined per action for a given table;

After Triggers:

Once again, as the name implies, once a DML action completes, the After Trigger executes. Here are some key features of After Triggers:

1) After Triggers are executed after a DML action, such as an Insert statement and any ensuing referential cascade actions and constraint checks have run;
2) A database action cannot be canceled using an After Trigger. This is due to the action having already been completed;
3) One or more After Triggers per action can be defined on a table, but having more than one adds to the complexity of code maintenance;
4) After triggers cannot be defined on database Views.

Special Database Objects Associated With Triggers

Triggers use two special database objects, “INSERTED” and “DELETED”, to access rows affected by the database actions. Within the scope of a trigger the “INSERTED” and “DELETED” objects have the identical columns as the affected Trigger’s table. These database objects can be referenced as Tables within the confines of the Trigger code logic.

The “INSERTED” table contains all the new values; whereas, the “DELETED” table contains old, removed values. Here is how these special database objects can be used:

1) An Insert Trigger - The “INSERTED” table determines which rows were added to the affected table;
2) A Delete Trigger - The “DELETED” table determines which rows were removed from the affected table;
3) An Update Trigger - Here, the “INSERTED” table is used to view the new or updated values of the affected table, and the “DELETED” table is used to view the values prior to the Update.

Create Trigger Statement

In the most simplified form, The basic SQL Server syntax for creating an Instead Of (Before) Trigger is as follows:

```
CREATE TRIGGER <InsteadOfTriggerName>
ON <TableName>
INSTEAD OF {[INSERT],[UPDATE],[DELETE]}  /* Either Insert, Update, or Delete
                                            specified */
AS
BEGIN
        <series sql code statements>
END;
```

In the most simplified form, The basic SQL Server syntax for creating an After Trigger is as follows:

```
CREATE TRIGGER <AfterTriggerName>
```

```
ON <TableName>
AFTER {[INSERT],[UPDATE],[DELETE]}   /* Either Insert, Update, or Delete specified */
AS
BEGIN
        <series sql code statements>
END;
```

NOTE: There are some subtle syntax differences for MySQL (and other database systems) when creating triggers, but the concept is identical. Please consult a MySQL reference manual for the subtle syntactical differences.

Trigger Examples

Example #1:

Using the 3-table example above, let us create an "After" Delete Trigger on Table Customer, so that if a record is deleted from the Customer table (that is, a CustomerId row is removed from the table), any Order table rows that contain that CustomerId are also deleted. This will ensure that database referential integrity is maintained, and that we will not have any extraneous rows (with non-existent customers) remaining in Table Order.

```
CREATE TRIGGER   AfterCustomerDeleteTrigger
ON               Customer
AFTER DELETE
AS
BEGIN
        DELETE FROM     Order
        WHERE DELETED.CustomerId = Order.CustomerId
END;
```

NOTE: This After Trigger is called whenever we try to delete one (or more) customers from Table Customer. We make use ot the Special Database Object "DELETED", to Obtain the Customer Id (or Customer Id's, if many customers are deleted), and to delete the requisite record(s) in Table Order, that contain that Customer Id (or Customer Id's, in the case of many customers).

Example #2:

Again, using our 3-table example above, let us create an "Instead Of" Insert Trigger on Table OrderItem, that checks to make sure that the OrderItemDescription is not already contained in the table. If it is, we rollback the Insert transaction. If it is not in the table yet, then we proceed with the insertion of a row into that table. Here is the definition and code for this Instead Of Trigger

```
CREATE TRIGGER   InsteadOfOrderItemInsertTrigger
ON               OrderItem
INSTEAD OF INSERT
AS
BEGIN
        DECLARE @OrderItemId              INT,
                @OrderItemDescription     VARCHAR(255)

        SELECT  @OrderItemId = INSERTED.OrderItemId,
                @OrderItemDescription = INSERTED.OrderItemDescription
        FROM    INSERTED

        IF (EXISTS(SELECT OrderItemDescription
                   FROM   OrderItem
                   WHERE  OrderItemDescription = @OrderItemDescription))
        BEGIN
                ROLLBACK
        END
        ELSE
                BEGIN
                        INSERT INTO OrderItem
                        VALUES (@OrderItemId, @OrderItemDescription)
                END
END;
```

NOTE: This Instead Of Trigger is called whenever we attempt to Insert a record into Table OrderItem. We make use ot the Special Database Object “INSERTED”, to obtain the attempted Insert of OrderItemId and OrderItemDescription. If OrderItemDescription already exists in Table OrderItem, then we Rollback the transaction (and refuse the Insert); otherwise, we Insert the record, which was the original intention before the Instead Of Trigger was executed.

Conclusion

SQL Triggers are valuable code segments that can be applied either before/after an Insert, Update, or Delete statement is executed on a given Table, to ensure that referential integrity is maintained in a database. Additionally, when Triggers are employed, the use of the “INSERTED” and “DELETED” special database objects can be used within the logic of the Trigger.