Lecture 9

Chapter 7 + 8: Triggers and Views

John Connor

November 7, 2018

Triggers

Triggers (7.5)

Three things are needed to create a *trigger*:

1. An **event**. Usually one or more of `INSERT`, `UPDATE`, `DELETE`.

Triggers (7.5)

Three things are needed to create a *trigger*:

1. An **event**. Usually one or more of `INSERT`, `UPDATE`, `DELETE`.
2. A **condition**. A condition to test, where a condition is essentially anything which can follow the `WHERE` symbol.

Triggers (7.5)

Three things are needed to create a *trigger*:

1. An **event**. Usually one or more of `INSERT`, `UPDATE`, `DELETE`.
2. A **condition**. A condition to test, where a condition is essentially anything which can follow the `WHERE` symbol.
3. An **action**. An action to preform whenever the event occures and the condition is met. The action usually takes the form of an `INSERT`, `UPDATE` or `DELETE`, but the action can also modify the query which triggered the event, or even *abort* the transaction which created the event.

Triggers (7.5)

Three things are needed to create a *trigger*:

1. An **event**. Usually one or more of `INSERT`, `UPDATE`, `DELETE`.
2. A **condition**. A condition to test, where a condition is essentially anything which can follow the `WHERE` symbol.
3. An **action**. An action to preform whenever the event occures and the condition is met. The action usually takes the form of an `INSERT`, `UPDATE` or `DELETE`, but the action can also modify the query which triggered the event, or even *abort* the transaction which created the event.

Triggers are sometimes called *event-condition-action* (ECA) rules.

Trigger Example (7.5.1)

```
CREATE TRIGGER NetWorthTrigger
AFTER UPDATE OF netWorth ON MovieExec
REFERENCING
      OLD ROW AS OldTuple
      NEW ROW AS NewTuple
FOR EACH ROW
WHEN (OldTuple.netWorth > NewTuple.netWorth)
      UPDATE MovieExec
      SET netWorth = OldTuple.netWorth
      WHERE cert# = NewTuple.cert#;
```

Trigger Details (7.5.1)

1. The condition is actually optional, without a `WHEN` clause, A trigger will always execute.
2. A trigger's check and action can be configured to take place either before or after the triggering event is executed. The relevant part of the syntax is `AFTER UPDATE`, `BEFORE UPDATE`.

Trigger Details (7.5.1)

1. The condition is actually optional, without a `WHEN` clause, A trigger will always execute.
2. A trigger's check and action can be configured to take place either before or after the triggering event is executed. The relevant part of the syntax is `AFTER UPDATE`, `BEFORE UPDATE`.
3. A trigger's event can be configured for specific attributes, such as `netWorth` in the previous example.

Trigger Details (7.5.1)

1. The condition is actually optional, without a `WHEN` clause, A trigger will always execute.
2. A trigger's check and action can be configured to take place either before or after the triggering event is executed. The relevant part of the syntax is `AFTER UPDATE`, `BEFORE UPDATE`.
3. A trigger's event can be configured for specific attributes, such as `netWorth` in the previous example.
   3.1 If the event is `INSERT` or `DELETE`, what does `OF netWorth` do?

Trigger Details (7.5.1)

1. The condition is actually optional, without a `WHEN` clause, A trigger will always execute.
2. A trigger's check and action can be configured to take place either before or after the triggering event is executed. The relevant part of the syntax is `AFTER UPDATE`, `BEFORE UPDATE`.
3. A trigger's event can be configured for specific attributes, such as `netWorth` in the previous example.
   3.1 If the event is `INSERT` or `DELETE`, what does `OF netWorth` do?
4. The trigger can be configured to execute for either

Trigger Details (7.5.1)

1. The condition is actually optional, without a `WHEN` clause, A trigger will always execute.
2. A trigger's check and action can be configured to take place either before or after the triggering event is executed. The relevant part of the syntax is `AFTER UPDATE`, `BEFORE UPDATE`.
3. A trigger's event can be configured for specific attributes, such as `netWorth` in the previous example.
   3.1 If the event is `INSERT` or `DELETE`, what does `OF netWorth` do?
4. The trigger can be configured to execute for either
   4.1 Once for each modified tuple (a row-level trigger).
   `FOR EACH ROW`

Trigger Details (7.5.1)

1. The condition is actually optional, without a `WHEN` clause, A trigger will always execute.
2. A trigger's check and action can be configured to take place either before or after the triggering event is executed. The relevant part of the syntax is `AFTER UPDATE`, `BEFORE UPDATE`.
3. A trigger's event can be configured for specific attributes, such as `netWorth` in the previous example.
   3.1 If the event is `INSERT` or `DELETE`, what does `OF netWorth` do?
4. The trigger can be configured to execute for either
   4.1 Once for each modified tuple (a row-level trigger).
   `FOR EACH ROW`
   4.2 Once for all of the tuples that are affected by the triggering event. (a statement-level trigger).
   `FOR EACH STATEMENT`

Trigger Details (7.5.1)

1. The condition is actually optional, without a `WHEN` clause, A trigger will always execute.
2. A trigger's check and action can be configured to take place either before or after the triggering event is executed. The relevant part of the syntax is `AFTER UPDATE`, `BEFORE UPDATE`.
3. A trigger's event can be configured for specific attributes, such as `netWorth` in the previous example.
   3.1 If the event is `INSERT` or `DELETE`, what does `OF netWorth` do?
4. The trigger can be configured to execute for either
   4.1 Once for each modified tuple (a row-level trigger).
   `FOR EACH ROW`
   4.2 Once for all of the tuples that are affected by the triggering event. (a statement-level trigger).
   `FOR EACH STATEMENT`
   What happens to `REFERENCING OLD ROW AS Foo`?

Trigger Details (7.5.1)

1. The condition is actually optional, without a `WHEN` clause, A trigger will always execute.
2. A trigger's check and action can be configured to take place either before or after the triggering event is executed. The relevant part of the syntax is `AFTER UPDATE`, `BEFORE UPDATE`.
3. A trigger's event can be configured for specific attributes, such as `netWorth` in the previous example.
   3.1 If the event is `INSERT` or `DELETE`, what does `OF netWorth` do?
4. The trigger can be configured to execute for either
   4.1 Once for each modified tuple (a row-level trigger).
   `FOR EACH ROW`
   4.2 Once for all of the tuples that are affected by the triggering event. (a statement-level trigger).
   `FOR EACH STATEMENT`
   What happens to `REFERENCING OLD ROW AS Foo`?
   It must become `REFERENCING OLD TABLE AS Foo`!

Trigger Happy

When should you use triggers?

Trigger Happy

When should you use triggers?

Here are some thoughts:

IBM Knowledge Center — When to use triggers

softwareengineering.stackexchange.com —
SQL Triggers and when or when not to use them

Can you spot any problems with the given advice?

The Most Common Use Of Triggers

```
CREATE TABLE Orders (
    ID INT PRIMARY KEY,
    Customer INT NOT NULL REFERENCES Customer(ID),
    Placed DATETIME NOT NULL,
    Approved DATETIME,
    Status CHAR(3)
);
```

```
CREATE TABLE OrderAudit (
    ID INT PRIMARY KEY AUTO_INCREMENT,
    Order INT REFERENCES Orders(ID),
    Customer INT REFERENCES Customer(ID),
    Placed DATETIME NOT NULL,
    Approved DATETIME,
    UpdatedOn DATETIME NOT NULL,
    UpdatedBy INT REFERENCES Users(ID)
);
```

The Most Common Use Of Triggers

```
CREATE TRIGGER OrderAudit
AFTER UPDATE, INSERT
REFERENCING
      NEW ROW AS NewTuple
FOR EACH ROW
INSERT INTO OrderAudit VALUES (
      0,
      NewTuple.ID,
      NewTuple.Customer,
      NewTuple.Placed,
      NewTuple.Approved,
      NewTuple.Status,
      CURRENT_TIMESTAMP,
      MagicFunctionToGetUserID()
);
```

Views

Views

A Table physically exists on disk, and its state is managed by the RDBMS.

Views

A Table physically exists on disk, and its state is managed by the RDBMS.

A View is a *virtual table*, a table that does necessarily exist on disk, and has no state.

Views

A Table physically exists on disk, and its state is managed by the RDBMS.

A View is a *virtual table*, a table that does necessarily exist on disk, and has no state.

In practice, a view is defined from a `SELECT` statement, but the result of the statement can be queried as if it were a table.

View Example (8.2)

Suppose we have the following relations:

`Movies(title, year, length, genre, studioName, producer#)`

`MovieExec(name, address, cert#, netWorth)`

View Example (8.2)

Suppose we have the following relations:

`Movies(title, year, length, genre, studioName, producer#)`

`MovieExec(name, address, cert#, netWorth)`

We can create a view which will associate to each movie the producer of the movie:

```
CREATE VIEW MovieProd AS
    SELECT title, name
      FROM Movies, MovieExec
     WHERE producer# = cert#;
```

View Example (8.2)

Suppose we have the following relations:

`Movies(title, year, length, genre, studioName, producer#)`

`MovieExec(name, address, cert#, netWorth)`

We can create a view which will associate to each movie the producer of the movie:

```
CREATE VIEW MovieProd AS
    SELECT title, name
      FROM Movies, MovieExec
     WHERE producer# = cert#;
```

A view can be dropped as if it were a table:

```
DROP VIEW MovieProd;
```

View Example (8.2)

Suppose we have the following relations:

`Movies(title, year, length, genre, studioName, producer#)`

`MovieExec(name, address, cert#, netWorth)`

We can create a view which will associate to each movie the producer of the movie:

```
CREATE VIEW MovieProd AS
    SELECT title, name
      FROM Movies, MovieExec
     WHERE producer# = cert#;
```

A view can be dropped as if it were a table:

```
DROP VIEW MovieProd;
```

Don't worry, this does not delete any data!

View Details

A view can be queried as if it were a table.

View Details

A view can be queried as if it were a table.

```
SELECT * FROM MovieProd;
```

View Details

A view can be queried as if it were a table.

```
SELECT * FROM MovieProd;
```

This is morally equivalent to

```
SELECT * FROM (
    SELECT title, name
      FROM Movies, MovieExec
     WHERE producer# = cert#
);
```

Updating Views

Can we update a view?

Updating Views

Can we update a view? Sometimes!

Updating Views

Can we update a view? Sometimes!
The rules are complicated! (And probably vary from RDBMS to RDBMS) According to the textbook

1. The view must be created with `SELECT`, not `SELECT DISTINCT`, and must only select attributes from a single relation, say R.
2. The `WHERE` clause must not reference R in a subquery.
3. The `FROM` clause can only consist of one occurrence of R and no other relation.
4. The list in the `SELECT` clause must specify all of the attributes which cannot be assigned default values.

Updating Views — Example

```
CREATE TABLE Example (
    Foo INT NOT NULL,
    Bar INT
);

CREATE VIEW ExampleView1 AS SELECT Foo FROM Example;

CREATE VIEW ExampleView2 AS SELECT Bar FROM Example;
```

Can either view be updated?

Updating Views — Example

```
CREATE TABLE Example (
    Foo INT NOT NULL,
    Bar INT
);
```

```
CREATE VIEW ExampleView1 AS SELECT Foo FROM Example;
```

```
CREATE VIEW ExampleView2 AS SELECT Bar FROM Example;
```

Can either view be updated?

ExampleView1 can be updated, while ExampleView2 cannot.

Updating Views With Triggers (8.8)

Is there a way around these restrictions?

Updating Views With Triggers (8.8)

Is there a way around these restrictions? Yes, there is a special type of trigger which can be defined for views.

```
CREATE TRIGGER ExampleTrigger
INSTEAD OF INSERT ON ExampleView2
REFERENCING NEW ROW as NewRow
FOR EACH ROW
INSERT INTO Example (Foo, Bar) VALUES (0, NewRow.Bar)
```