Need a datetime field in MS SQL that automatically updates when the record is modified

20

5

I need to create a new DATETIME field in MS-SQL that will always contain the date of when the record was created, and then it needs to automatically update whenever the record is modified. I've heard people say I need a trigger, which is fine, but I don't know how to write it. Could somebody help with the syntax for a trigger to accomplish this?

In MySQL terms, it should do exactly the same as this MySQL statement:

```
ADD `modstamp` timestamp NULL DEFAULT CURRENT_TIMESTAMP on
update CURRENT_TIMESTAMP
```

Here are a few requirements:

- I can't alter my UPDATE statements to set the field when the row is modified, because I don't control the application logic that writes to the records.
- Ideally, I would not need to know the names of any other columns in the table (such as the primary key)
- It should be short and efficient, because it will happen very often.

MSSQL, actually, don't have a way to define a default value for UPDATE.

So you need to add a column with default value for inserting:

```
ADD modstamp DATETIME2 NULL DEFAULT GETDATE()
```

And add a Trigger on the table:

```
CREATE TRIGGER tgr_modstamp
ON **TABLENAME**
AFTER UPDATE AS
  UPDATE **TABLENAME**
  SET ModStamp = GETDATE()
  WHERE **ID** IN (SELECT DISTINCT **ID** FROM Inserted)
```

And yes, you need to specify a identity column for each trigger.

CAUTION: take care when inserting columns on tables where you don't know the code of the application. If your app have INSERT VALUES command without field definition, it will raise errors even with default value on new columns.

12

Okay, I always like to keep track of not only when something happened but who did it!

Lets create a test table in [tempdb] named [dwarfs]. At a prior job, a financial institution, we keep track of inserted (create) date and updated (modify) date.

```
-- just playing
use tempdb;
go

-- drop table
if object_id('dwarfs') > 0
drop table dwarfs
go

-- create table
create table dwarfs
(
asigned_id int identity(1,1),
full_name varchar(16),
ins_date datetime,
ins_name sysname,
upd_date datetime,
upd_name sysname,
);
go

-- insert/update dates
alter table dwarfs
    add constraint [df_ins_date] default (getdate()) for
ins_date;
alter table dwarfs
    add constraint [df_upd_date] default (getdate()) for
upd_date;

-- insert/update names
alter table dwarfs
```

```
    add constraint [df_ins_name] default
(coalesce(suser_sname(),'?')) for ins_name;

alter table dwarfs
    add constraint [df_upd_name] default
(coalesce(suser_sname(),'?')) for upd_name;
go
```

For updates, but the inserted and deleted tables exist. I choose to join on the inserted for the update.

```
-- create the update trigger
create trigger trg_changed_info on dbo.dwarfs
for update
as
begin

    -- nothing to do?
    if (@@rowcount = 0)
      return;

    update d
    set
       upd_date = getdate(),
       upd_name = (coalesce(suser_sname(),'?'))
    from
       dwarfs d join inserted i
    on
       d.asigned_id = i.asigned_id;

end
go
```

Last but not least, lets test the code. Anyone can type a untested TSQL statement in. However, I always stress testing to my team!

```
-- remove data
truncate table dwarfs;
go

-- add data
insert into dwarfs (full_name) values
('bilbo baggins'),
('gandalf the grey');
```

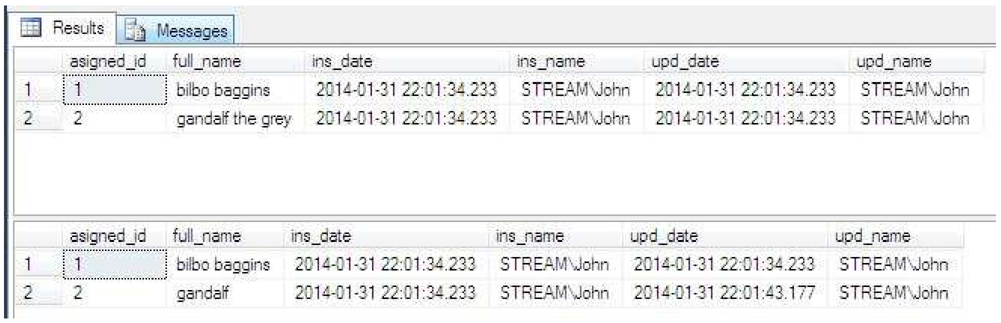
```
go

-- show the data
select * from dwarfs;

-- update data
update dwarfs
set full_name = 'gandalf'
where asigned_id = 2;

-- show the data
select * from dwarfs;
```

The output. I only waited 10 seconds between the insert and the delete. Nice thing is that who and when are both captured.



```
   asigned_id  full_name        ins_date                 ins_name      upd_date                 upd_name
1  1           bilbo baggins    2014-01-31 22:01:34.233  STREAM\John   2014-01-31 22:01:34.233  STREAM\John
2  2           gandalf the grey 2014-01-31 22:01:34.233  STREAM\John   2014-01-31 22:01:34.233  STREAM\John

   asigned_id  full_name        ins_date                 ins_name      upd_date                 upd_name
1  1           bilbo baggins    2014-01-31 22:01:34.233  STREAM\John   2014-01-31 22:01:34.233  STREAM\John
2  2           gandalf          2014-01-31 22:01:34.233  STREAM\John   2014-01-31 22:01:43.177  STREAM\John
```

This is possible since SQL Server 2016 by using `PERIOD FOR SYSTEM_TIME`.

This is something that was introduced for temporal tables but you don't have to use temporal tables to use this.

An example is below

```
CREATE TABLE dbo.YourTable
(
    FooId INT PRIMARY KEY CLUSTERED,
    FooName VARCHAR(50) NOT NULL,
    modstamp DATETIME2 GENERATED ALWAYS AS ROW START NOT NULL,
    MaxDateTime2 DATETIME2 GENERATED ALWAYS AS ROW END HIDDEN
NOT NULL,
    PERIOD FOR SYSTEM_TIME (modstamp,MaxDateTime2)
)

INSERT INTO dbo.YourTable (FooId, FooName)
VALUES      (1,'abc');

SELECT *
```

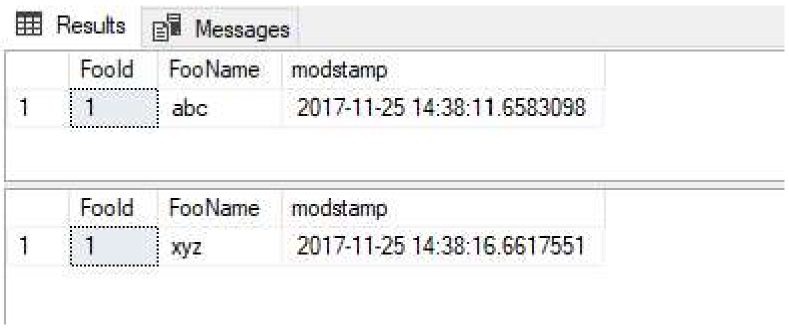
```
FROM   dbo.YourTable;

WAITFOR DELAY '00:00:05'

UPDATE dbo.YourTable
SET    FooName = 'xyz'
WHERE  FooId = 1;

SELECT *
FROM   dbo.YourTable;

DROP TABLE dbo.YourTable;
```



```
   FooId  FooName  modstamp
1  1      abc      2017-11-25 14:38:11.6583098

   FooId  FooName  modstamp
1  1      xyz      2017-11-25 14:38:16.6617551
```

It has some limitations.

- The time stored will be updated by the system and always be UTC.
- There is a need to declare a second column (`MaxDateTime2` above) that is completely superfluous for this use case. But it can be marked as hidden making it easier to ignore.