**Exercise 13 – Triggers to log changes - InnoDB**

Logging changes is a vital function of any database system. Often the logging is handled by the client application however there may be dozens of applications working with the database happily inserting, updating and deleting away and some of those may not be logging these changes. You can add a trigger to log any changes to a table in the background so no one, not even the developers writing client applications will know the changes are being logged, the log table may be in the same database, or another database on the server or even on another server completely.

In this exercise we will create another database on the same server and log changes to it

Let us get started

1. We need to create a logging database and table, open a new SQL Query tab, enter the following script and execute it. A new database named **logging\_database** will appear in your database list

DROP SCHEMA IF EXISTS `logging\_database`;

CREATE DATABASE `logging\_database` /\*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8\_unicode\_ci \*/ /\*!80016 DEFAULT ENCRYPTION='N' \*/;

CREATE TABLE `logging\_database`.`changelog` (

`ID` INT NOT NULL AUTO\_INCREMENT,

`Database` VARCHAR(45) NOT NULL,

`TableName` VARCHAR(45) NOT NULL,

`PKValue` VARCHAR(45) NOT NULL,

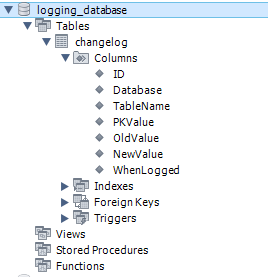
`OldValue` VARCHAR(255) NOT NULL,

`NewValue` VARCHAR(255) NOT NULL,

`WhenLogged` TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

PRIMARY KEY (`ID`)

);



**Figure 4.50 New logging\_database with one table**

The logging table is very minimal and will hold the basic information we want to demonstrate. OldValue and NewValue are 255 characters long which will work for most fields however if your real-world database has long text fields, blobs or any other large datatypes, you will need to cater for those as well, and would also need to consider, do you really want to log such field types? A logging table can gather a huge number of entries very quickly on a busy database.

1. Now that we have a logging database, we need to create the trigger to log any changes to the members table. Open a new SQL Query tab and enter the following script. It’s a long script but as with all scripts in the course, it is available in the system resources in **Chapter04/Exercise13** named **Create members after update log Trigger.txt**, copy and paste it in ☺ We will explain the script later, for now just execute it.

USE `autoclub`;

DELIMITER $$

DROP TRIGGER IF EXISTS autoclub.LogMemberChanges$$

USE `autoclub`$$

CREATE TRIGGER `LogMemberChanges` AFTER UPDATE ON `members` FOR EACH ROW BEGIN

if NEW.Surname <> OLD.Surname THEN

INSERT INTO `logging\_database`.`changelog` (`Database`, `TableName`, `PKValue`, `OldValue`, `NewValue`)

VALUES ('autoclub', 'Members',OLD.ID, OLD.Surname,NEW.Surname);

end if;

if NEW.MiddleNames <> OLD.MiddleNames THEN

INSERT INTO `logging\_database`.`changelog` (`Database`, `TableName`, `PKValue`, `OldValue`, `NewValue`)

VALUES ('autoclub', 'Members',OLD.ID, OLD.MiddleNames,NEW.MiddleNames);

end if;

if NEW.FirstName <> OLD.FirstName THEN

INSERT INTO `logging\_database`.`changelog` (`Database`, `TableName`, `PKValue`, `OldValue`, `NewValue`)

VALUES ('autoclub', 'Members',OLD.ID, OLD.FirstName,NEW.FirstName);

end if;

if NEW.DOB <> OLD.DOB THEN

INSERT INTO `logging\_database`.`changelog` (`Database`, `TableName`, `PKValue`, `OldValue`, `NewValue`)

VALUES ('autoclub', 'Members',OLD.ID, OLD.DOB ,NEW.DOB );

end if;

if NEW.PhotoPath <> OLD.PhotoPath THEN

INSERT INTO `logging\_database`.`changelog` (`Database`, `TableName`, `PKValue`, `OldValue`, `NewValue`)

VALUES ('autoclub', 'Members',OLD.ID, OLD.PhotoPath,NEW.PhotoPath);

end if;

if NEW.SigPath <> OLD.SigPath THEN

INSERT INTO `logging\_database`.`changelog` (`Database`, `TableName`, `PKValue`, `OldValue`, `NewValue`)

VALUES ('autoclub', 'Members',OLD.ID, OLD.SigPath,NEW.SigPath);

end if;

if NEW.Active <> OLD.Active THEN

INSERT INTO `logging\_database`.`changelog` (`Database`, `TableName`, `PKValue`, `OldValue`, `NewValue`)

VALUES ('autoclub', 'Members',OLD.ID, OLD.Active,NEW.Active);

end if;

if NEW.JoinDate <> OLD.JoinDate THEN

INSERT INTO `logging\_database`.`changelog` (`Database`, `TableName`, `PKValue`, `OldValue`, `NewValue`)

VALUES ('autoclub', 'Members',OLD.ID, OLD.JoinDate,NEW.JoinDate);

end if;

if NEW.InactiveDate <> OLD.InactiveDate THEN

INSERT INTO `logging\_database`.`changelog` (`Database`, `TableName`, `PKValue`, `OldValue`, `NewValue`)

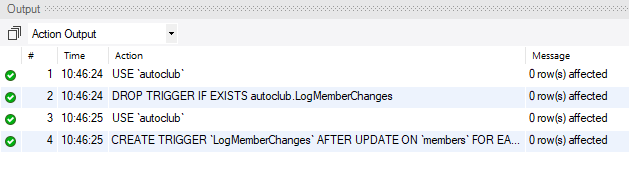
VALUES ('autoclub', 'Members',OLD.ID, OLD.InactiveDate,NEW.InactiveDate);

end if;

END $$

DELIMITER ;

After execution, the output window should display similar to below



**Figure 4.51 Successful execution of the Create Trigger script**

1. So let us test this trigger, we will start with a good test first where nothing is going to stop it updating. Open a new SQL Query tab and enter the following script and run it.

UPDATE `autoclub`.`members` SET `MiddleNames`='Xavier' WHERE `ID`='1';

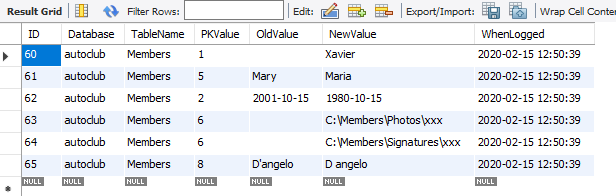
UPDATE `autoclub`.`members` SET `FirstName`='Maria' WHERE `ID`='5';

UPDATE `autoclub`.`members` SET `DOB`='1980-10-15' WHERE `ID`='2';

UPDATE `autoclub`.`members` SET `PhotoPath`='C:\\Members\\Photos\\xxx', `SigPath`='C:\\Members\\Signatures\\xxx' WHERE `ID`='6';

UPDATE `autoclub`.`members` SET `MiddleNames`='D angelo' WHERE `ID`='8';

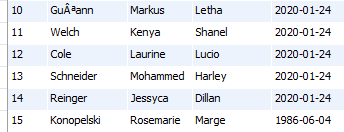
1. Now go to the **logging\_database** and Select All Rows on the **changelog** table, the results will be



**Figure 4.52 All changes have been logged showing OLD and NEW values**

So that was pretty good, all the changes were logged and we didn’t have to do a thing

1. Now we see how the transactional engine handles an age check fails in the script, open a new SQL Query tab and enter the following script, everything is valid except the third line will fail the age test. Run it. The image immediately shows the original values and the script below that shows the new values to be updated.



**Figure 4.53 Original values of records 10 to 15**

UPDATE `autoclub`.`members` SET `DOB`='2000-01-24' WHERE `ID`='10';

UPDATE `autoclub`.`members` SET `DOB`='2001-01-24' WHERE `ID`='11';

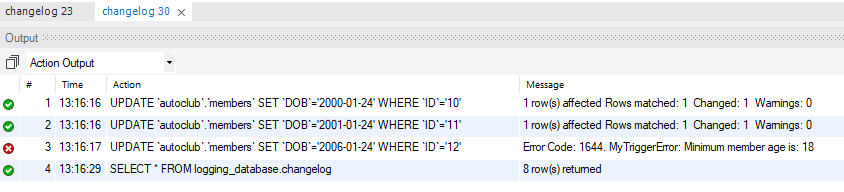
UPDATE `autoclub`.`members` SET `DOB`='2006-01-24' WHERE `ID`='12';

UPDATE `autoclub`.`members` SET `DOB`='2002-01-24' WHERE `ID`='13';

UPDATE `autoclub`.`members` SET `DOB`='2003-01-24' WHERE `ID`='14';

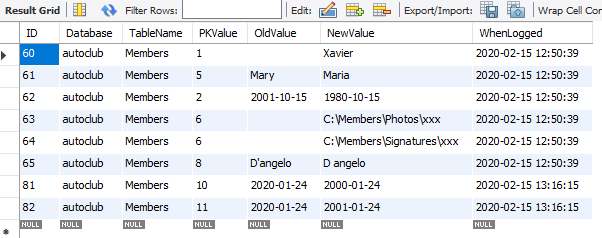
UPDATE `autoclub`.`members` SET `DOB`='2004-06-04' WHERE `ID`='15';

1. When the script is run from the Query tab, the first two records are updated and the third fails the age check causing an error.



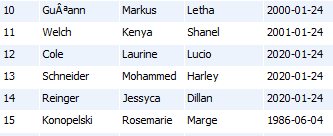
**Figure 4.54 Output from the test script**

1. The log file shows two entries were changed, rows 81 and 82 (I have done much testing so don’t worry about the ID is out of order)



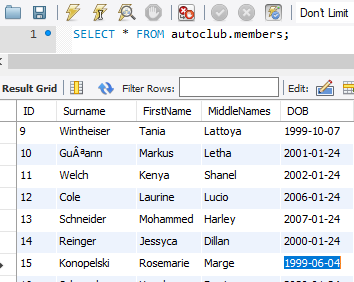
**Figure 4.55 Two changes were recorded in the log table**

1. And when we examine the records again (rerun the windows query to reload the table data), only two were changed, all records after the failure did not get processed.



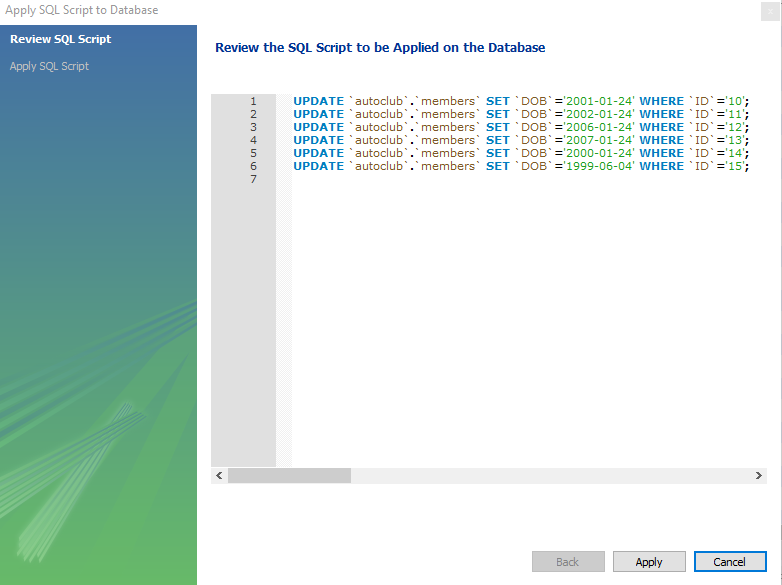
**Figure 4.56 Only two records were updated in the members table**

1. If you modify the data directly onscreen you will get different results. Now that the data window is open, change the values exactly as shown below for DOB, directly on the screen as shown below for records 10 to 15, be sure to click Apply when done



**Figure 4.57 Change the records onscreen as shown here for DOB and click Apply**

1. After making the changes and clicking Apply, the following screen will open

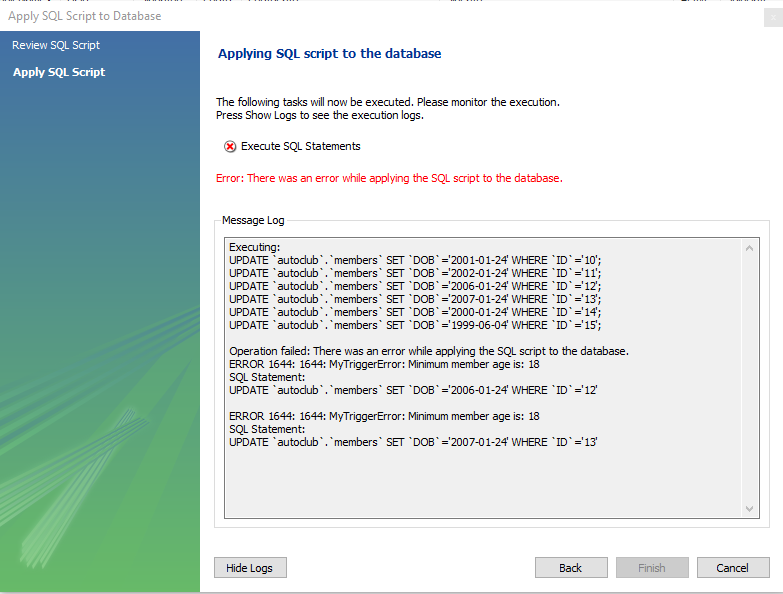


**Figure 4.58 Review SQL script screen**

Note

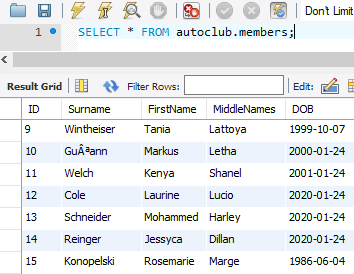
If you copy and paste the above SQL into a SQL Query tab, you will get the same results as in the previous example (steps 5 to 8), DO NOT do this here, we are demonstrating the behaviour of this method of updating so continue on. Perhaps, copy the text above and put it aside for now, after we are finished this exercise, run it and see what happens.

1. Click apply, the following window will open



**Figure 4.59 Execution of the SQL displays the error**

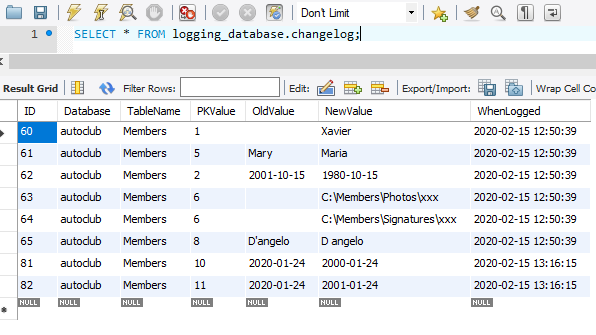
1. Now here is where it is different, return to the members data screen and click the Lightning Bolt to refresh the screen, locate records 10 to 15.



**Figure 4.60 Record values have returned to their values prior to your changes**

Nothing has changed, the entire transaction was rolled back because a single record failed the age test.

1. Review the records in the Change Log, nothing has been recorded.



**Figure 4.61 nothing was recorded in the changelog**

**The conclusion to Exercise 13 – Triggers to log changes - InnoDB**

AFTER UPDATE logging is pretty cool, it works well without having to specifically code it (after the trigger is created that is) but would you really want to do it, what if you had hundreds of tables, would you want to create the trigger for all of them? I prefer a nice dynamic little routine in my applications that logs changes. Triggers also do not allow dynamic field names for the NEW and OLD commands so you have to write code for every field you want to log.

**Exercise 14 – Triggers to log changes – MyISAM**In this exercise we will mainly be going through the processes of testing how the triggers behave when the table is using the MyISAM Non-Transactional engine. All tables in the **autoclub** database are using the InnoDB Transactional engine so we need to change the engine on the **members** table so we can run the same tests, normally you would open the table using the **Alter Table** option in Workbench and change the engine however, we cannot because there are Foreign Keys set on other tables referencing the members table and MySQL will not allow the engine to be changed whilst there are Foreign Keys referencing it, but we have a plan. To get around this, we are going to run a single script that will

* copy the members table to a new table called **members\_MyISAM** specifically for testing
* set the Primary Key and Auto-Increment properties
* change the new tables engine to MyISAM, we can do this because nothing is referencing the new table in FK’s
* create the BEFORE UPDATE age check trigger, renamed to **CheckMemberAge\_MyISAM**
* create the AFTER UPDATE logging trigger, renamed to **LogMemberChanges\_ MyISAM**

So let us get started with this test

1. Start by opening a new SQL Query tab and running the following script. The script text is in a file located in the Chapter4/Exercise14 folder in the course resources so you can copy/paste the script, the file is named **Prepare MyISAM version of members table.txt**

USE `autoclub`;

-- Copy the members table to members\_MyISAM

DROP TABLE IF EXISTS autoclub.members\_MyISAM;

CREATE TABLE IF NOT EXISTS members\_MyISAM SELECT \* FROM

members;

-- Set the Primary Key field and the Auto Increment properies on ID

ALTER TABLE `autoclub`.`members\_MyISAM`

CHANGE COLUMN `ID` `ID` INT(11) NOT NULL AUTO\_INCREMENT ,

ADD PRIMARY KEY (`ID`);

-- Change the tables engine to MyISAM so we can test it

ALTER TABLE `autoclub`.`members\_MyISAM`

ENGINE = MyISAM ;

-- Create the trigger for members\_MyISAM

DELIMITER $$

DROP TRIGGER IF EXISTS autoclub.CheckMemberAge\_MyISAM$$

USE `autoclub`$$

CREATE TRIGGER `CheckMemberAge\_MyISAM` BEFORE UPDATE ON `members\_MyISAM` FOR EACH ROW BEGIN

declare msg varchar(128);

SET @MinAge = (SELECT `Value` FROM LOOKUPS WHERE `KEY`='MinMemberAge');

if NEW.dob > (SELECT DATE\_SUB(curdate(), interval @MinAge year)) THEN

set msg = concat('MyTriggerError: Minimum member age is: ', @MinAge);

signal sqlstate '45000' set message\_text = msg;

end if;

END$$

DROP TRIGGER IF EXISTS autoclub.LogMemberChanges\_MyISAM$$

CREATE TRIGGER `LogMemberChanges\_MyISAM` AFTER UPDATE ON `members\_MyISAM` FOR EACH ROW BEGIN

if NEW.Surname <> OLD.Surname THEN

INSERT INTO `logging\_database`.`changelog` (`Database`, `TableName`, `PKValue`, `OldValue`, `NewValue`)

VALUES ('autoclub', 'Members\_MyISAM',OLD.ID, OLD.Surname,NEW.Surname);

end if;

if NEW.MiddleNames <> OLD.MiddleNames THEN

INSERT INTO `logging\_database`.`changelog` (`Database`, `TableName`, `PKValue`, `OldValue`, `NewValue`)

VALUES ('autoclub', 'Members\_MyISAM',OLD.ID, OLD.MiddleNames,NEW.MiddleNames);

end if;

if NEW.FirstName <> OLD.FirstName THEN

INSERT INTO `logging\_database`.`changelog` (`Database`, `TableName`, `PKValue`, `OldValue`, `NewValue`)

VALUES ('autoclub', 'Members\_MyISAM',OLD.ID, OLD.FirstName,NEW.FirstName);

end if;

if NEW.DOB <> OLD.DOB THEN

INSERT INTO `logging\_database`.`changelog` (`Database`, `TableName`, `PKValue`, `OldValue`, `NewValue`)

VALUES ('autoclub', 'Members\_MyISAM',OLD.ID, OLD.DOB ,NEW.DOB );

end if;

if NEW.PhotoPath <> OLD.PhotoPath THEN

INSERT INTO `logging\_database`.`changelog` (`Database`, `TableName`, `PKValue`, `OldValue`, `NewValue`)

VALUES ('autoclub', 'Members\_MyISAM',OLD.ID, OLD.PhotoPath,NEW.PhotoPath);

end if;

if NEW.SigPath <> OLD.SigPath THEN

INSERT INTO `logging\_database`.`changelog` (`Database`, `TableName`, `PKValue`, `OldValue`, `NewValue`)

VALUES ('autoclub', 'Members\_MyISAM',OLD.ID, OLD.SigPath,NEW.SigPath);

end if;

if NEW.Active <> OLD.Active THEN

INSERT INTO `logging\_database`.`changelog` (`Database`, `TableName`, `PKValue`, `OldValue`, `NewValue`)

VALUES ('autoclub', 'Members\_MyISAM',OLD.ID, OLD.Active,NEW.Active);

end if;

if NEW.JoinDate <> OLD.JoinDate THEN

INSERT INTO `logging\_database`.`changelog` (`Database`, `TableName`, `PKValue`, `OldValue`, `NewValue`)

VALUES ('autoclub', 'Members\_MyISAM',OLD.ID, OLD.JoinDate,NEW.JoinDate);

end if;

if NEW.InactiveDate <> OLD.InactiveDate THEN

INSERT INTO `logging\_database`.`changelog` (`Database`, `TableName`, `PKValue`, `OldValue`, `NewValue`)

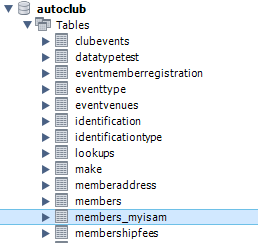
VALUES ('autoclub', 'Members\_MyISAM',OLD.ID, OLD.InactiveDate,NEW.InactiveDate);

end if;

END $$

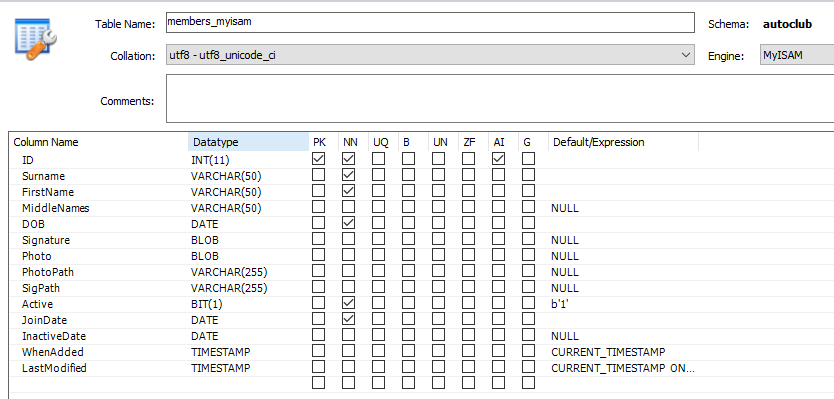
DELIMITER ;

After running the above script, the new table will appear in the tables list, after you refresh the list of course



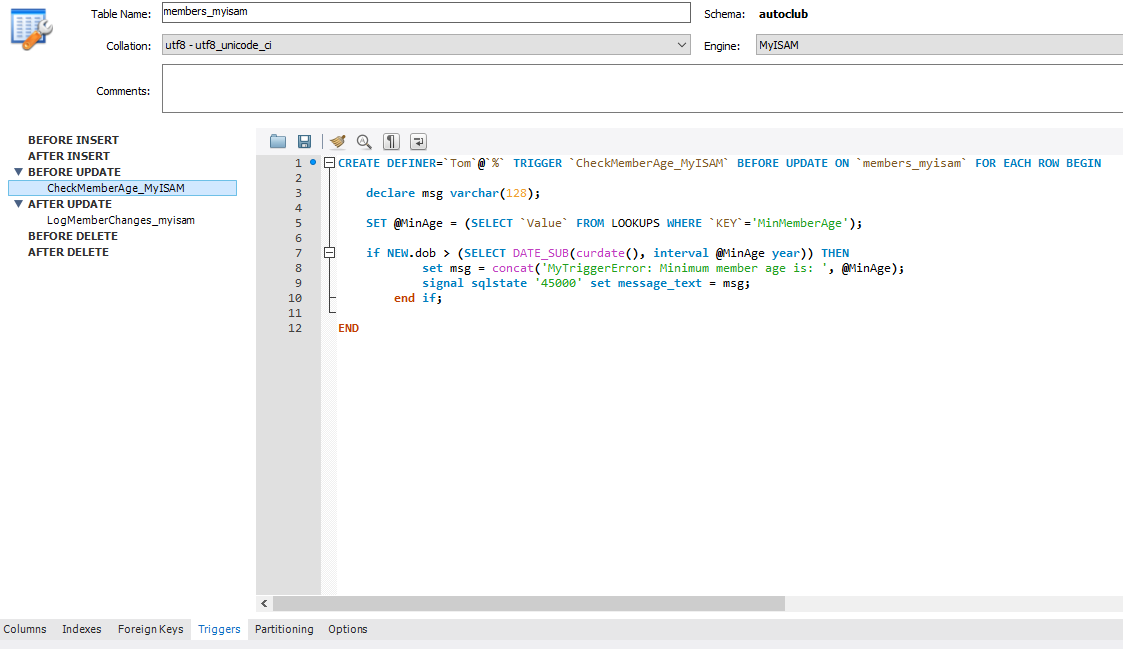
**Figure 4.62 The new table will appear in the tables list**

And if you view the table with Alter Table you will see it is identical to the members table in structure and the engine is MyISAM

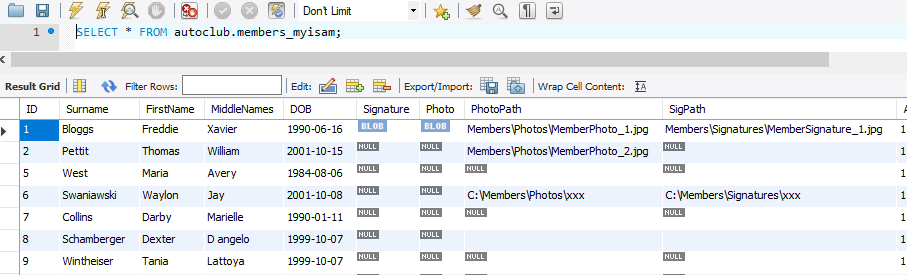


**Figure 4.63 Identical to the members table in structure and the engine is MyISAM**

Click the Triggers tab, you will see the triggers have been created, both refer to the new members\_myisam table



**Figure 4.64 Triggers have been created**



**Figure 4.65 The data has been copied over as well**

We are now set to run the tests.

1. First, the good data test, open a new SQL Query tab, enter the following script and execute it

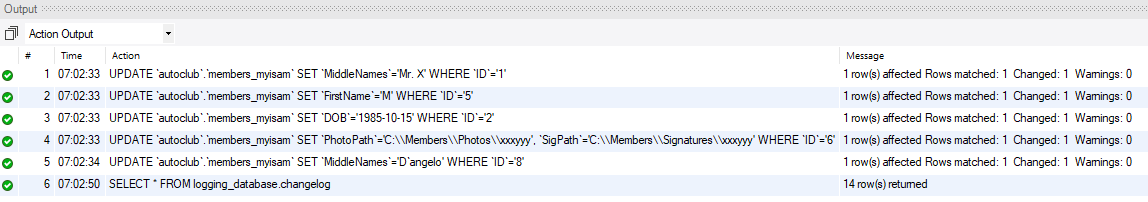
UPDATE `autoclub`.`members\_myisam` SET `MiddleNames`='Mr. X' WHERE `ID`='1';

UPDATE `autoclub`.`members\_myisam` SET `FirstName`='M' WHERE `ID`='5';

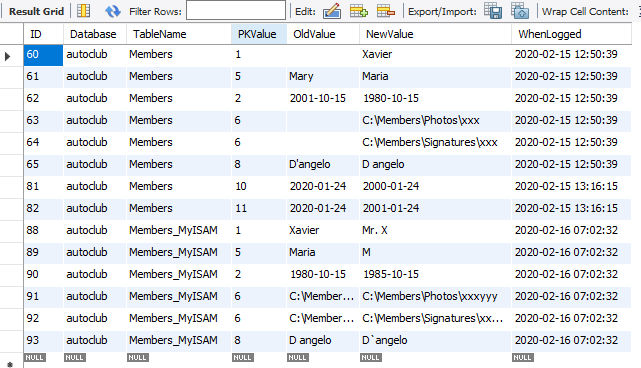
UPDATE `autoclub`.`members\_myisam` SET `DOB`='1985-10-15' WHERE `ID`='2';

UPDATE `autoclub`.`members\_myisam` SET `PhotoPath`='C:\\Members\\Photos\\xxxyyy', `SigPath`='C:\\Members\\Signatures\\xxxyyy' WHERE `ID`='6';

UPDATE `autoclub`.`members\_myisam` SET `MiddleNames`='D`angelo' WHERE `ID`='8';

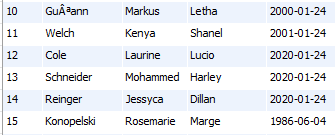


**Figure 4.66 All updates worked as expected**



**Figure 4.67 All updates logged**

1. The current DOB values for records 10 to 15 are



**Figure 4.68 Current DOB values for records 10 to 15**

1. Open a new SQL Query tab and run the following script. This script will fail the age check on record ID 12, the third line.

UPDATE `autoclub`.`members\_myisam` SET `DOB`='1995-01-24' WHERE `ID`='10';

UPDATE `autoclub`.`members\_myisam` SET `DOB`='1996-01-24' WHERE `ID`='11';

UPDATE `autoclub`.`members\_myisam` SET `DOB`='2006-01-24' WHERE `ID`='12';

UPDATE `autoclub`.`members\_myisam` SET `DOB`='2002-01-24' WHERE `ID`='13';

UPDATE `autoclub`.`members\_myisam` SET `DOB`='2003-01-24' WHERE `ID`='14';

UPDATE `autoclub`.`members\_myisam` SET `DOB`='2004-06-04' WHERE `ID`='15';

When run the following output will show, failed as expected on the third line



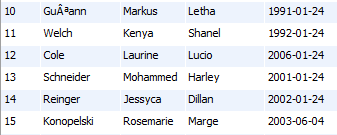
**Figure 4.69 Failed the age check on the third line, ID 12**

The two successful updates were logged



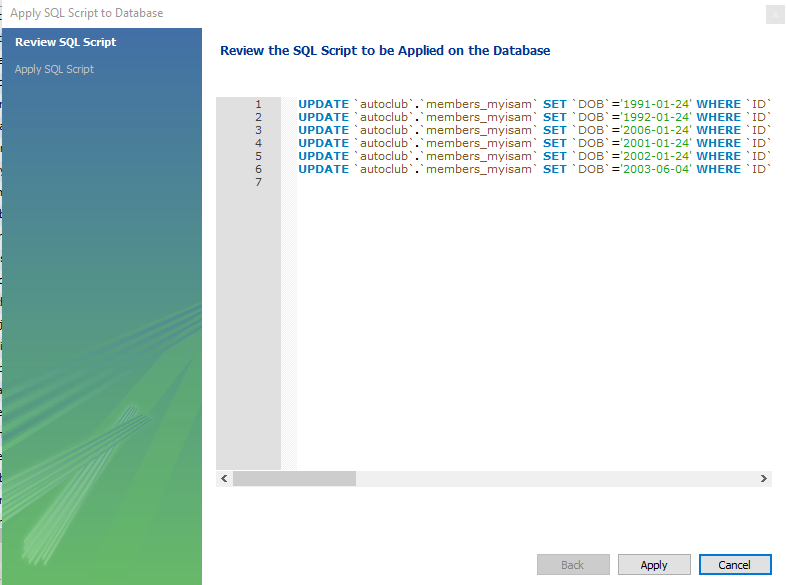
**Figure 4.70 Successful log for first two record updates**

1. Now make the changes to the **DOB** fields directly in the **members\_myisam** data screen (Right-click on the table and Select All Rows) in Workbench exactly as shown below, again line 3 will fail the age check, and click **Apply**

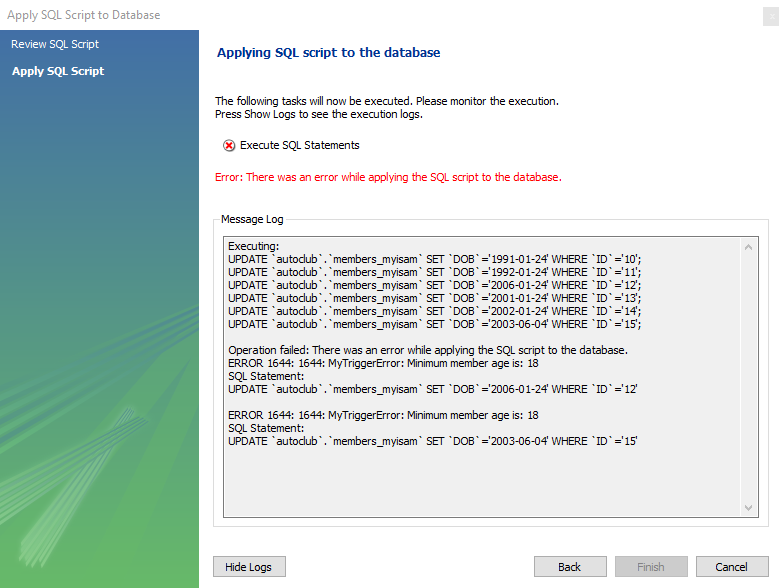


**Figure 4.71 Changes directly in the Workbench table data view**

1. Same script, different values but will be run in the wizard, click Apply

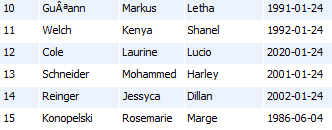


**Figure 4.72 Script designed to fail age check on record three**



**Figure 4.73 Failed as expected**

1. Now check the data window by refreshing it and examine the results on records 10 to 15



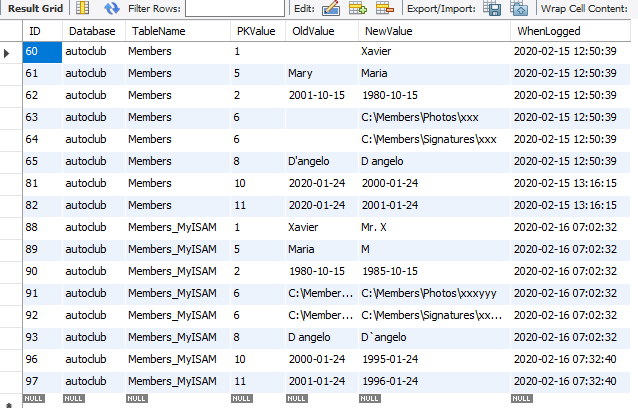
Records 10 and 11 were updated in the table as expected.

Record 12 failed, that was expected also.

Records 13 and 14 were updated, it was expected that anything after 12 would not have been updated

Record 15 was not updated, with 13 and 14 being updated, I would have expected this be updated as well.

And finally, check the log table, nothing was logged



**Figure 4.74 No new log entries**

**The conclusion to Exercise 14 – Triggers to log changes - MyISAM**

The MyISAM Engine AFTER UPDATE logging works the same as the InnoDB, both work well. The engines do work differently when it comes to the BEFORE UPDATE data validation when there is a fail on the Age Check and the **signal sqlstate '45000'** is executed to cancel the update. The table below is for easier comparison of the Exercises 13 and 14’s results.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Script Description** | **InnoDB** | **MyISAM** |
| **1** | Query tab SQL, all valid data | All records updated, all updates logged | All records updated, all updates logged |
| **2** | Query tab, age check fail line 3 | First two records updated and logged, remaining records not updated or logged. | First two records updated and logged, remaining records not updated or logged |
| **3** | Workbench Data screen direct changes, age check fail on record 12 (records are 10 to 15) | * The entire transaction was rolled back on age check fail on record 12. * Nothing was updated or logged. | * Records 10 and 11 were updated in the table as expected. * Record 12 failed as expected * Records 13 and 14 were updated, it was expected that anything after 12 would not have been updated * Record 15 was not updated, with 13 and 14 being updated, I would have expected this be updated as well. * **The updates were not logged** |

Test 1, worked as expected for both engines

Test 2, worked as expected for MyISAM, however, the InnoDB Transactional Engine should have rolled back the entire transaction on the raising of the error, see note.

Test 3, InnoDB worked as expected, see note. MyISAM had strange results. Records 10 and 11 were updated, that was expected and 12 failed the age check, the statement should have stopped at this point however it updated two more records and not the last. The statement was in fact halted because the changes that were made, were not logged. This is a major issue as it means the if the AFTER UPDATE trigger was completing some kind of calculations or adding other records (other than our logging) then this could result in incorrect or incomplete data.

Note

The following is a direct quote from the MySQL 8.0 Reference Manual, 24.3.1 Trigger Syntax and Examples  
For transactional tables, failure of a statement should cause rollback of all changes performed by the statement. Failure of a trigger causes the statement to fail, so trigger failure also causes rollback. For nontransactional tables, such rollback cannot be done, so although the statement fails, any changes performed prior to the point of the error remain in effect.

Time for an activity, in the next activity you will be creating a trigger to log changes to the membershipfees table

**Activity 5 – Log membership fee payments**One area of a business database that requires logging is anything to with finances. You have been asked to automate the logging of the payment of membership fees, this is urgent as the member will start paying the fees week, it an annual thing. The developer of the Admin System has gone overseas and has not left anyone to look after the system in his absence, management has made a note to speak to him about this but nonetheless, you need set up the auditing and you cannot modify the Admin application, how do you do it? With a trigger as a temporary solution until our wayward developer returns in two months.

In this activity, you will

* Create a log table in the **logging\_database** to store the fee logs, the fields you need to record are
  + MemberID
  + FeeAmount
  + DatePaid
* Create a trigger to log all record inserts on the **membershipfees** table with the above details
* No BEFORE INSERT checks are required for this activity due to inconsistencies
* The trigger should be on the AFTER INSERT event
* Test the logging with a single record insert
* Test the logging with a multi-record insert, two or more

**The solution to Activity 5 – Log membership fee payments**

CREATE TABLE `logging\_database`.`membershipfee\_payments` (

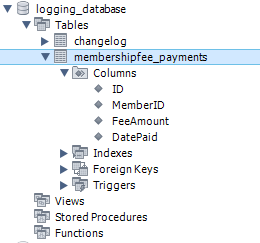
`ID` INT NOT NULL AUTO\_INCREMENT,

`MemberID` INT NOT NULL,

`FeeAmount` DECIMAL(10,2) NOT NULL,

`DatePaid` DATE NOT NULL,

PRIMARY KEY (`ID`));



**Figure 4.75 membershipfee\_ payments table**

USE `autoclub`;

DELIMITER $$

DROP TRIGGER IF EXISTS autoclub.LogMemberShipFeePayments$$

USE `autoclub`$$

CREATE TRIGGER `LogMemberShipFeePayments` AFTER INSERT ON `MemberShipFees` FOR EACH ROW

BEGIN

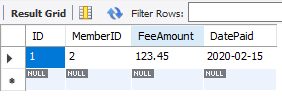
INSERT INTO `logging\_database`.`membershipfee\_payments` (MemberID, FeeAmount, DatePaid)

VALUES (new.MemberID, new.FeeAmount, new.DatePaid);

END $$

DELIMITER ;

INSERT INTO `autoclub`.`membershipfees` (`MemberID`,`FeeAmount`,`DatePaid`) VALUES (2,123.45,'2020-02-15');

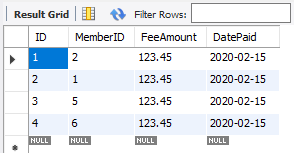


**Figure 4.76 Logged result of a single transaction**

INSERT INTO `autoclub`.`membershipfees` (`MemberID`,`FeeAmount`,`DatePaid`) VALUES (1,123.45,'2020-02-15');

INSERT INTO `autoclub`.`membershipfees` (`MemberID`,`FeeAmount`,`DatePaid`) VALUES (5,123.45,'2020-02-15');

INSERT INTO `autoclub`.`membershipfees` (`MemberID`,`FeeAmount`,`DatePaid`) VALUES (6,123.45,'2020-02-15');



**Figure 4.77 Logged result of multiple transactions**

Note

There are no member records with ID’s of 3 and 4 which is why they were not included in the above script. As there is a Foreign Key relationship between the **membersfees** and the **members** table with a RESTRICT constraint, an attempt to insert a record for ID 3 or 4 in the **membersfees** will be rejected by the FK constraint. Try it and check out the FK constraints in action

INSERT INTO `autoclub`.`membershipfees` (`MemberID`,`FeeAmount`,`DatePaid`) VALUES (3,123.45,'2020-02-15');