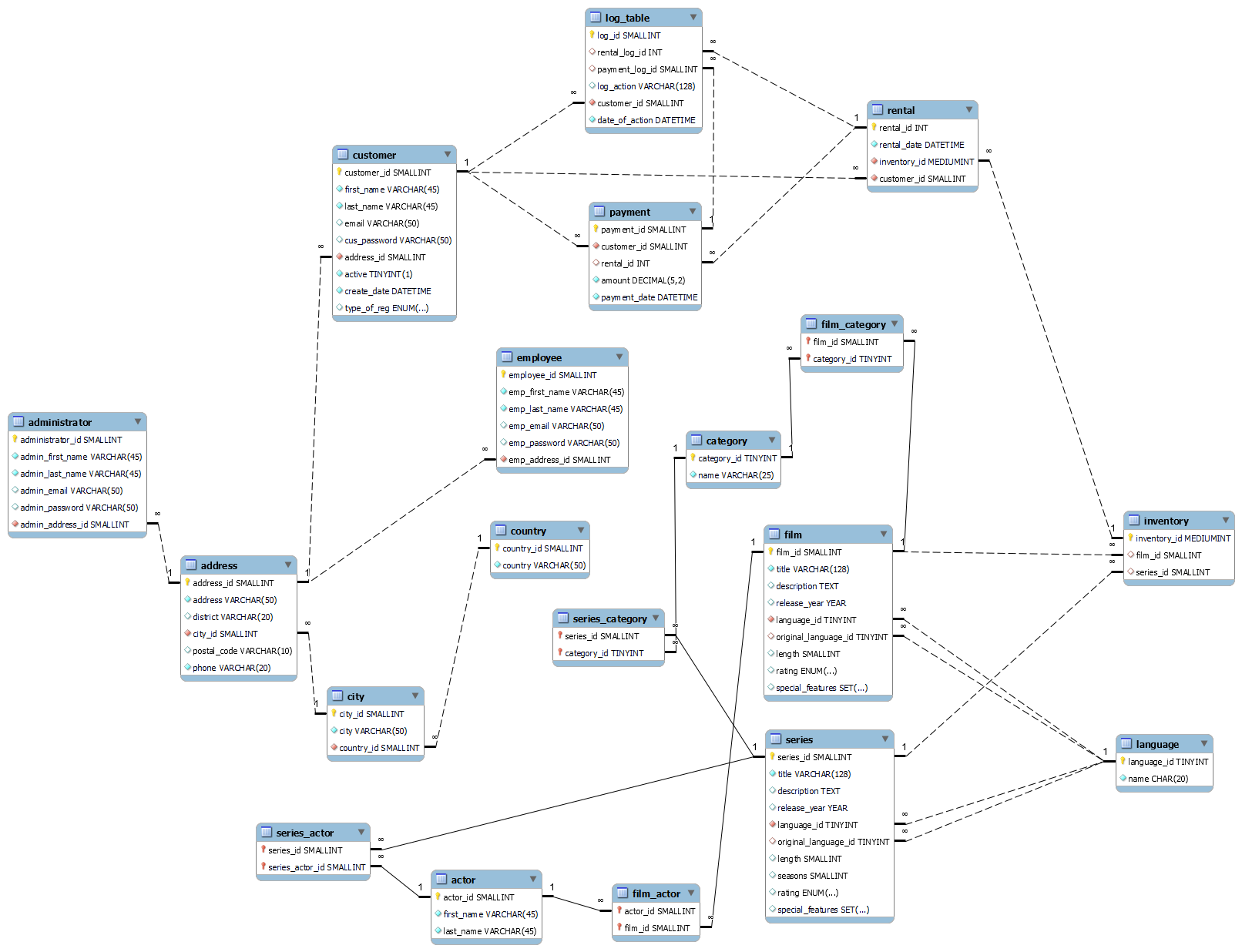
***Αναφορά Πρότζεκτ Βάσεων Δεδομένων - 2021-22***

Ονόματα Συμμετεχόντων:

1. Κωνσταντίνος Παρασκευόπουλος, ΑΜ: 1072608 , email: [st1072608@ceid.upatras.gr](mailto:st1067517@ceid.upatras.gr) , [up1072608@upnet.gr](mailto:up1067517@upnet.gr)
2. Βλάσιος Παναγιώτης Παναγιώτου , ΑΜ: 1067517 , email: [st1067517@ceid.upatras.gr](mailto:st1067517@ceid.upatras.gr) , [up1067517@upnet.gr](mailto:up1067517@upnet.gr)

***Μέρος Α: Σχεδιασμός ΒΔ και SQL***

***Κεφάλαιο 1: Σχολιασμός του αναθεωρημένου σχεσιακού σχήματος της ΒΔ***

**Α)** 

**Β)** Ο πίνακας **administrator** είναι πίνακας ο οποίος περιλαμβάνει τους διαχειριστές του συστήματος.

Ο πίνακας **employee** περιέχει τους υπαλλήλους της υπηρεσίας

στους δύο αυτούς πίνακες, και στον πίνακα customer, έχει προστεθεί πεδίο το οποίο περιέχει τον κωδικό της αντίστοιχης οντότητας (customer, employee ή administrator).

Στον αρχικό πίνακα **customer** έχει προστεθεί το πεδίο type\_of\_reg το οποίο είναι τύπου ENUM και μπορεί να πάρει τις τιμές MOVIES, SERIES, SERIES AND MOVIES.

Επίσης προσθέσαμε , με βάση τις οδηγίες της εκφώνησης , τους πίνακες **series**, **series\_category** και **series\_actor**. Αυτοί οι πίνακες δημιουργήθηκαν με βάση τους αντίστοιχους Film-πίνακες.

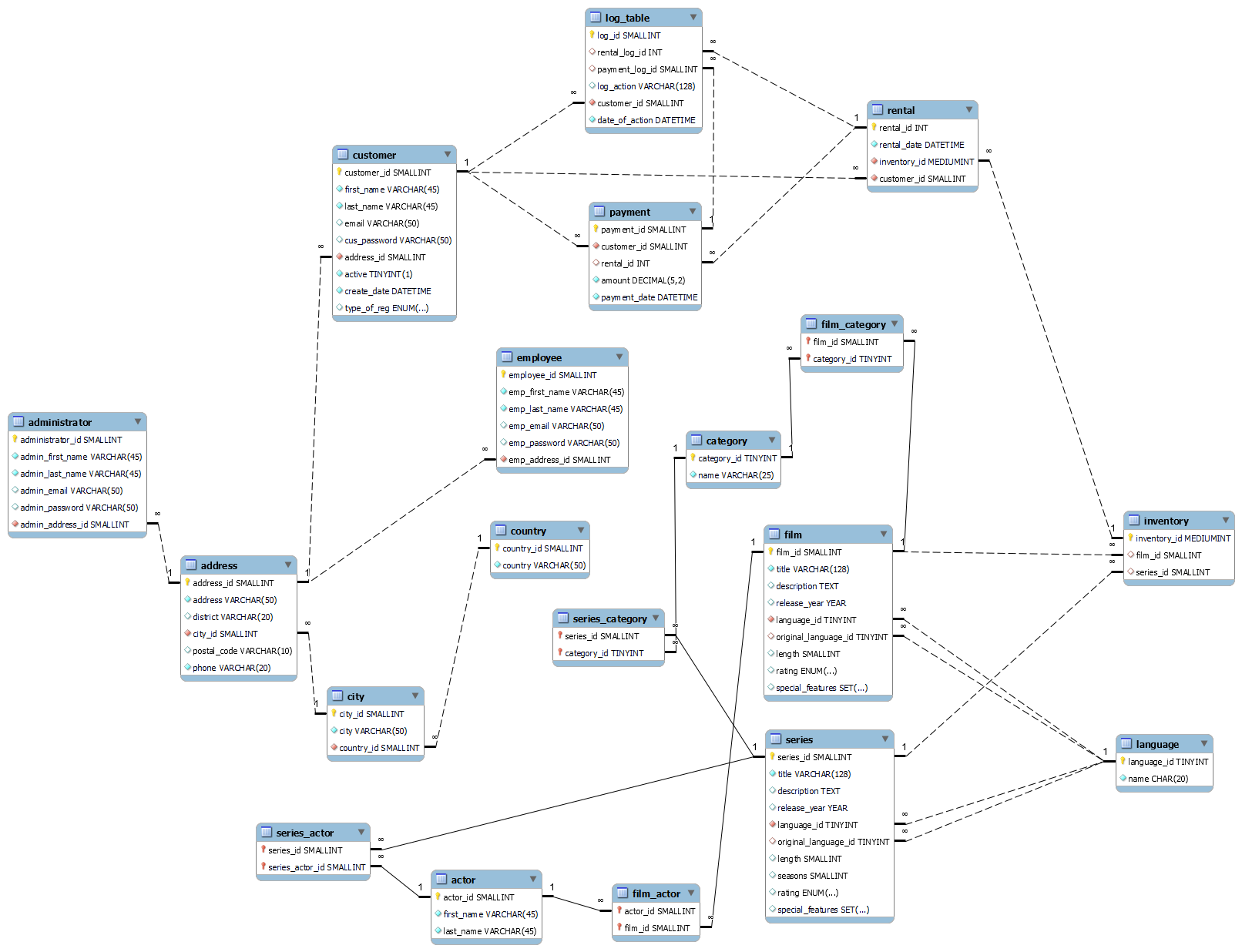
Για τη συνύπαρξη πινάκων ταινιών και πινάκων σειρών επεξεργαστήκαμε και τροποποιήσαμε τον πίνακα **inventory**, προσθέτοντας το πεδίο SeriesID και δίνοντας τη δυνατότητα στο πεδίο αυτό και στο πεδίο filmID να πάρουν την τιμή NULL , και πετύχαμε τη συνύπαρξη αυτή.

Συγκεκριμένα όταν το ένα πεδίο παίρνει την τιμή NULL το άλλο πεδίο παίρνει την αντίστοιχη τιμή και έτσι μπορούμε να γνωρίζουμε μέσω του inventory αν το περιεχόμενο πρόκειται για σειρά ή ταινία.

Επίσης. προσθέσαμε τον πίνακα **log\_table** στον οποίο αποθηκεύονται οι ενέργειες (INSERT, DELETE, UPDATE) που έγιναν στους πίνακες **payment** και **rental,** καθώς και το πεδίο rentalID της αντίστοιχης ενοικίασης, το πεδίο paymentID της αντίστοιχης πληρωμής, το πεδίο customerID του πελάτη που πραγματοποίησε την αντίστοιχη πληρωμή ή ενοικίαση, καθώς και ημερομηνία και ώρα όπου πραγματοποιήθηκε η κάθε ενέργεια.

Για τα πεδία rental\_log\_id και payment\_log\_id ισχύει αντίστοιχη λογική με τον πίνακα inventory, όπου το καθένα παίρνει την τιμή που πρέπει και το άλλο παίρνει την τιμή NULL ώστε να γνωρίζουμε αν η ενέργεια που έγινε, έγινε στον πίνακα rental ή στον πίνακα payment.

Οι προαναφερθέντες πίνακες φαίνονται και στο ακόλουθο σχεσιακό διάγραμμα:



***Κεφάλαιο 2: Create - Insert***

Ο κώδικας των create για την δημιουργία των πινάκων της ΒΔ είναι ο εξής:

-- Database TVOnDemand

-- based on

-- Sakila Sample Database Schema

-- Version 1.2

-- Copyright (c) 2006, 2019, Oracle and/or its affiliates.

-- Redistribution and use in source and binary forms, with or without

-- modification, are permitted provided that the following conditions are

-- met:

-- \* Redistributions of source code must retain the above copyright notice,

-- this list of conditions and the following disclaimer.

-- \* Redistributions in binary form must reproduce the above copyright

-- notice, this list of conditions and the following disclaimer in the

-- documentation and/or other materials provided with the distribution.

-- \* Neither the name of Oracle nor the names of its contributors may be used

-- to endorse or promote products derived from this software without

-- specific prior written permission.

-- THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS

-- IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,

-- THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR

-- PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR

-- CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,

-- EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,

-- PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR

-- PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF

-- LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING

-- NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS

-- SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

DROP SCHEMA IF EXISTS tvondemand;

CREATE SCHEMA tvondemand;

USE tvondemand;

--

-- Table structure for table `actor`

--

CREATE TABLE actor (

actor\_id SMALLINT UNSIGNED NOT NULL AUTO\_INCREMENT,

first\_name VARCHAR(45) NOT NULL,

last\_name VARCHAR(45) NOT NULL,

PRIMARY KEY (actor\_id)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

--

-- Table structure for table `country`

--

CREATE TABLE country (

country\_id SMALLINT UNSIGNED NOT NULL AUTO\_INCREMENT,

country VARCHAR(50) NOT NULL,

PRIMARY KEY (country\_id)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

--

-- Table structure for table `city`

--

CREATE TABLE city (

city\_id SMALLINT UNSIGNED NOT NULL AUTO\_INCREMENT,

city VARCHAR(50) NOT NULL,

country\_id SMALLINT UNSIGNED NOT NULL,

PRIMARY KEY (city\_id),

CONSTRAINT `fk\_city\_country` FOREIGN KEY (country\_id) REFERENCES country (country\_id) ON DELETE RESTRICT ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

--

-- Table structure for table `address`

--

CREATE TABLE address (

address\_id SMALLINT UNSIGNED NOT NULL AUTO\_INCREMENT,

address VARCHAR(50) NOT NULL,

district VARCHAR(20) DEFAULT NULL,

city\_id SMALLINT UNSIGNED NOT NULL,

postal\_code VARCHAR(10) DEFAULT NULL,

phone VARCHAR(20) NOT NULL,

PRIMARY KEY (address\_id),

CONSTRAINT `fk\_address\_city` FOREIGN KEY (city\_id) REFERENCES city (city\_id) ON DELETE RESTRICT ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

--

-- Table structure for table `category`

--

CREATE TABLE category (

category\_id TINYINT UNSIGNED NOT NULL AUTO\_INCREMENT,

name VARCHAR(25) NOT NULL,

PRIMARY KEY (category\_id)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

--

-- Table structure for table `language`

--

CREATE TABLE language (

language\_id TINYINT UNSIGNED NOT NULL AUTO\_INCREMENT,

name CHAR(20) NOT NULL,

PRIMARY KEY (language\_id)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

--

-- Table structure for table `customer`

--

CREATE TABLE customer (

customer\_id SMALLINT UNSIGNED NOT NULL AUTO\_INCREMENT,

first\_name VARCHAR(45) NOT NULL,

last\_name VARCHAR(45) NOT NULL,

email VARCHAR(50) DEFAULT NULL,

cus\_password VARCHAR(50) DEFAULT NULL,

address\_id SMALLINT UNSIGNED NOT NULL,

active BOOLEAN NOT NULL DEFAULT TRUE,

create\_date DATETIME NOT NULL,

type\_of\_reg ENUM('MOVIES','SERIES','SERIES AND MOVIES'),

PRIMARY KEY (customer\_id),

CONSTRAINT fk\_customer\_address FOREIGN KEY (address\_id) REFERENCES address (address\_id) ON DELETE RESTRICT ON UPDATE CASCADE

)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

--

-- Table structure for table `employee`

-- ADDED 19/1/2022

CREATE TABLE employee (

employee\_id SMALLINT UNSIGNED NOT NULL AUTO\_INCREMENT,

emp\_first\_name VARCHAR(45) NOT NULL,

emp\_last\_name VARCHAR(45) NOT NULL,

emp\_email VARCHAR(50) DEFAULT NULL,

emp\_password VARCHAR(50) DEFAULT NULL,

emp\_address\_id SMALLINT UNSIGNED NOT NULL,

PRIMARY KEY (employee\_id),

CONSTRAINT fk\_emp\_address FOREIGN KEY (emp\_address\_id) REFERENCES address (address\_id) ON DELETE RESTRICT ON UPDATE CASCADE

);

--

-- Table structure for table `administrator`

-- ADDED 19/1/2022

CREATE TABLE administrator (

administrator\_id SMALLINT UNSIGNED NOT NULL AUTO\_INCREMENT,

admin\_first\_name VARCHAR(45) NOT NULL,

admin\_last\_name VARCHAR(45) NOT NULL,

admin\_email VARCHAR(50) DEFAULT NULL,

admin\_password VARCHAR(50) DEFAULT NULL,

admin\_address\_id SMALLINT UNSIGNED NOT NULL,

PRIMARY KEY (administrator\_id),

CONSTRAINT fk\_admin\_address FOREIGN KEY (admin\_address\_id) REFERENCES address (address\_id) ON DELETE RESTRICT ON UPDATE CASCADE

);

--

-- Table structure for table `film`

--

CREATE TABLE film (

film\_id SMALLINT UNSIGNED NOT NULL AUTO\_INCREMENT,

title VARCHAR(128) NOT NULL,

description TEXT DEFAULT NULL,

release\_year YEAR DEFAULT NULL,

language\_id TINYINT UNSIGNED NOT NULL,

original\_language\_id TINYINT UNSIGNED DEFAULT NULL,

length SMALLINT UNSIGNED DEFAULT NULL,

rating ENUM('G','PG','PG-13','R','NC-17') DEFAULT 'G',

special\_features SET('Trailers','Commentaries','Deleted Scenes','Behind the Scenes') DEFAULT NULL,

PRIMARY KEY (film\_id),

CONSTRAINT fk\_film\_language FOREIGN KEY (language\_id) REFERENCES language (language\_id) ON DELETE RESTRICT ON UPDATE CASCADE,

CONSTRAINT fk\_film\_language\_original FOREIGN KEY (original\_language\_id) REFERENCES language (language\_id) ON DELETE RESTRICT ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

--

-- Table structure for `series`

-- ADDED 19/1/2022\*/

CREATE TABLE series (

series\_id SMALLINT UNSIGNED NOT NULL AUTO\_INCREMENT,

title VARCHAR(128) NOT NULL,

description TEXT DEFAULT NULL,

release\_year YEAR DEFAULT NULL,

language\_id TINYINT UNSIGNED NOT NULL,

original\_language\_id TINYINT UNSIGNED DEFAULT NULL,

length SMALLINT UNSIGNED DEFAULT NULL,

seasons SMALLINT(2) UNSIGNED DEFAULT 0,

rating ENUM('G','PG','PG-13','R','NC-17') DEFAULT 'G',

special\_features SET('Trailers','Commentaries','Deleted Scenes','Behind the Scenes') DEFAULT NULL,

PRIMARY KEY (series\_id),

CONSTRAINT fk\_series\_language FOREIGN KEY (language\_id) REFERENCES language (language\_id) ON DELETE RESTRICT ON UPDATE CASCADE,

CONSTRAINT fk\_series\_language\_original FOREIGN KEY (original\_language\_id) REFERENCES language (language\_id) ON DELETE RESTRICT ON UPDATE CASCADE

);

--

-- Table structure for table `film\_actor`

--

CREATE TABLE film\_actor (

actor\_id SMALLINT UNSIGNED NOT NULL,

film\_id SMALLINT UNSIGNED NOT NULL,

PRIMARY KEY (actor\_id,film\_id),

CONSTRAINT fk\_film\_actor\_actor FOREIGN KEY (actor\_id) REFERENCES actor (actor\_id) ON DELETE RESTRICT ON UPDATE CASCADE,

CONSTRAINT fk\_film\_actor\_film FOREIGN KEY (film\_id) REFERENCES film (film\_id) ON DELETE RESTRICT ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

--

-- Table structure for table `film\_category`

--

CREATE TABLE film\_category (

film\_id SMALLINT UNSIGNED NOT NULL,

category\_id TINYINT UNSIGNED NOT NULL,

PRIMARY KEY (film\_id, category\_id),

CONSTRAINT fk\_film\_category\_film FOREIGN KEY (film\_id) REFERENCES film (film\_id) ON DELETE RESTRICT ON UPDATE CASCADE,

CONSTRAINT fk\_film\_category\_category FOREIGN KEY (category\_id) REFERENCES category (category\_id) ON DELETE RESTRICT ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

--

-- Table structure for table `series\_actor`

-- ADDED 19/1/2022

CREATE TABLE series\_actor (

series\_id SMALLINT UNSIGNED NOT NULL,

series\_actor\_id SMALLINT UNSIGNED NOT NULL,

PRIMARY KEY (series\_actor\_id, series\_id),

CONSTRAINT fk\_series\_actor\_actor FOREIGN KEY (series\_actor\_id) REFERENCES actor (actor\_id) ON DELETE RESTRICT ON UPDATE CASCADE,

CONSTRAINT fk\_series\_actor\_series FOREIGN KEY (series\_id) REFERENCES series (series\_id) ON DELETE RESTRICT ON UPDATE CASCADE

);

--

-- Table structure for table `series\_category`

-- ADDED 19/1/2022

CREATE TABLE series\_category (

series\_id SMALLINT UNSIGNED NOT NULL,

category\_id TINYINT UNSIGNED NOT NULL,

PRIMARY KEY (series\_id, category\_id),

CONSTRAINT fk\_series\_category\_series FOREIGN KEY (series\_id) REFERENCES series (series\_id) ON DELETE RESTRICT ON UPDATE CASCADE,

CONSTRAINT fk\_series\_category\_category FOREIGN KEY (category\_id) REFERENCES category (category\_id) ON DELETE RESTRICT ON UPDATE CASCADE

);

--

-- Table structure for table `inventory`

--

CREATE TABLE inventory (

inventory\_id MEDIUMINT UNSIGNED NOT NULL AUTO\_INCREMENT,

film\_id SMALLINT UNSIGNED NULL,

series\_id SMALLINT UNSIGNED NULL,

PRIMARY KEY (inventory\_id),

CONSTRAINT fk\_inventory\_film FOREIGN KEY (film\_id) REFERENCES film (film\_id) ON DELETE RESTRICT ON UPDATE CASCADE,

CONSTRAINT fk\_inventory\_series FOREIGN KEY (series\_id) REFERENCES series (series\_id) ON DELETE RESTRICT ON UPDATE CASCADE

);

--

-- Table structure for table `rental`

--

CREATE TABLE rental (

rental\_id INT NOT NULL AUTO\_INCREMENT,

rental\_date DATETIME NOT NULL,

inventory\_id MEDIUMINT UNSIGNED NOT NULL,

customer\_id SMALLINT UNSIGNED NOT NULL,

PRIMARY KEY (rental\_id),

UNIQUE KEY (rental\_date,inventory\_id,customer\_id),

CONSTRAINT fk\_rental\_inventory FOREIGN KEY (inventory\_id) REFERENCES inventory (inventory\_id) ON DELETE RESTRICT ON UPDATE CASCADE,

CONSTRAINT fk\_rental\_customer FOREIGN KEY (customer\_id) REFERENCES customer (customer\_id) ON DELETE RESTRICT ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

--

-- Table structure for table `payment`

--

CREATE TABLE payment (

payment\_id SMALLINT UNSIGNED NOT NULL AUTO\_INCREMENT,

customer\_id SMALLINT UNSIGNED NOT NULL,

rental\_id INT DEFAULT NULL,

amount DECIMAL(5,2) NOT NULL,

payment\_date DATETIME NOT NULL,

PRIMARY KEY (payment\_id),

CONSTRAINT fk\_payment\_rental FOREIGN KEY (rental\_id) REFERENCES rental (rental\_id) ON DELETE SET NULL ON UPDATE CASCADE,

CONSTRAINT fk\_payment\_customer FOREIGN KEY (customer\_id) REFERENCES customer (customer\_id) ON DELETE RESTRICT ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE log\_table (

log\_id SMALLINT UNSIGNED NOT NULL AUTO\_INCREMENT,

rental\_log\_id INT ,

payment\_log\_id SMALLINT UNSIGNED ,

log\_action varchar(128) DEFAULT NULL,

customer\_id smallint unsigned NOT NULL,

date\_of\_action datetime NOT NULL,

PRIMARY KEY (log\_id),

CONSTRAINT fk\_rental\_log FOREIGN KEY (rental\_log\_id) REFERENCES rental (rental\_id) ON DELETE SET NULL ON UPDATE CASCADE,

CONSTRAINT fk\_customer\_log FOREIGN KEY (customer\_id) REFERENCES customer (customer\_id) ON DELETE RESTRICT ON UPDATE CASCADE,

CONSTRAINT fk\_payment\_log FOREIGN KEY (payment\_log\_id) REFERENCES payment (payment\_id) ON DELETE SET NULL ON UPDATE CASCADE

);

Προσθέσαμε , με βάση τις οδηγίες της εκφώνησης , τους πίνακες **series**, **series\_category** και **series\_actor**. Αυτοί οι πίνακες δημιουργήθηκαν με βάση τους αντίστοιχους Film-πίνακες.

Επίσης. προσθέσαμε τον πίνακα **log\_table** στον οποίο αποθηκεύονται οι ενέργειες (INSERT, DELETE, UPDATE) που έγιναν στους πίνακες **payment** και **rental,** καθώς και το πεδίο rentalID της αντίστοιχης ενοικίασης, το πεδίο paymentID της αντίστοιχης πληρωμής, το πεδίο customerID του πελάτη που πραγματοποίησε την αντίστοιχη πληρωμή ή ενοικίαση, καθώς και ημερομηνία και ώρα όπου πραγματοποιήθηκε η κάθε ενέργεια.

Για τα πεδία rental\_log\_id και payment\_log\_id ισχύει αντίστοιχη λογική με τον πίνακα inventory, όπου το καθένα παίρνει την τιμή που πρέπει και το άλλο παίρνει την τιμή NULL ώστε να γνωρίζουμε αν η ενέργεια που έγινε, έγινε στον πίνακα rental ή στον πίνακα payment

Ο κώδικας των insert για την εισαγωγή δεδομένων στους πίνακες της ΒΔ δίνεται στο ZIP αρχείο διότι ήταν πάνω από 12.000 γραμμές.

***Κεφάλαιο 3: Stored Procedures***

**3.1**

DELIMITER $

DROP PROCEDURE IF EXISTS most\_rentals\_during\_this\_period$

CREATE PROCEDURE most\_rentals\_during\_this\_period (IN type\_of\_product CHAR(1) , IN quantity SMALLINT(3), IN date1 DATETIME, IN date2 DATETIME)

BEGIN

IF (type\_of\_product = 'm') THEN

SELECT count(\*), film.title , film.film\_id

FROM film

INNER JOIN inventory ON inventory.film\_id=film.film\_id

INNER JOIN rental ON rental.inventory\_id=inventory.inventory\_id

WHERE rental.rental\_date >= date1 AND rental.rental\_date <= date2

GROUP BY rental.inventory\_id

ORDER BY count(\*) DESC LIMIT quantity;

ELSEIF (type\_of\_product = 's') THEN

SELECT count(\*), series.title , series.series\_id

FROM series

INNER JOIN inventory ON inventory.series\_id=series.series\_id

INNER JOIN rental ON rental.inventory\_id=inventory.inventory\_id

WHERE rental.rental\_date > date1 AND rental.rental\_date < date2

GROUP BY rental.inventory\_id

ORDER BY count(\*) DESC LIMIT quantity;

ELSE

SELECT 'WRONG PARAMETER';

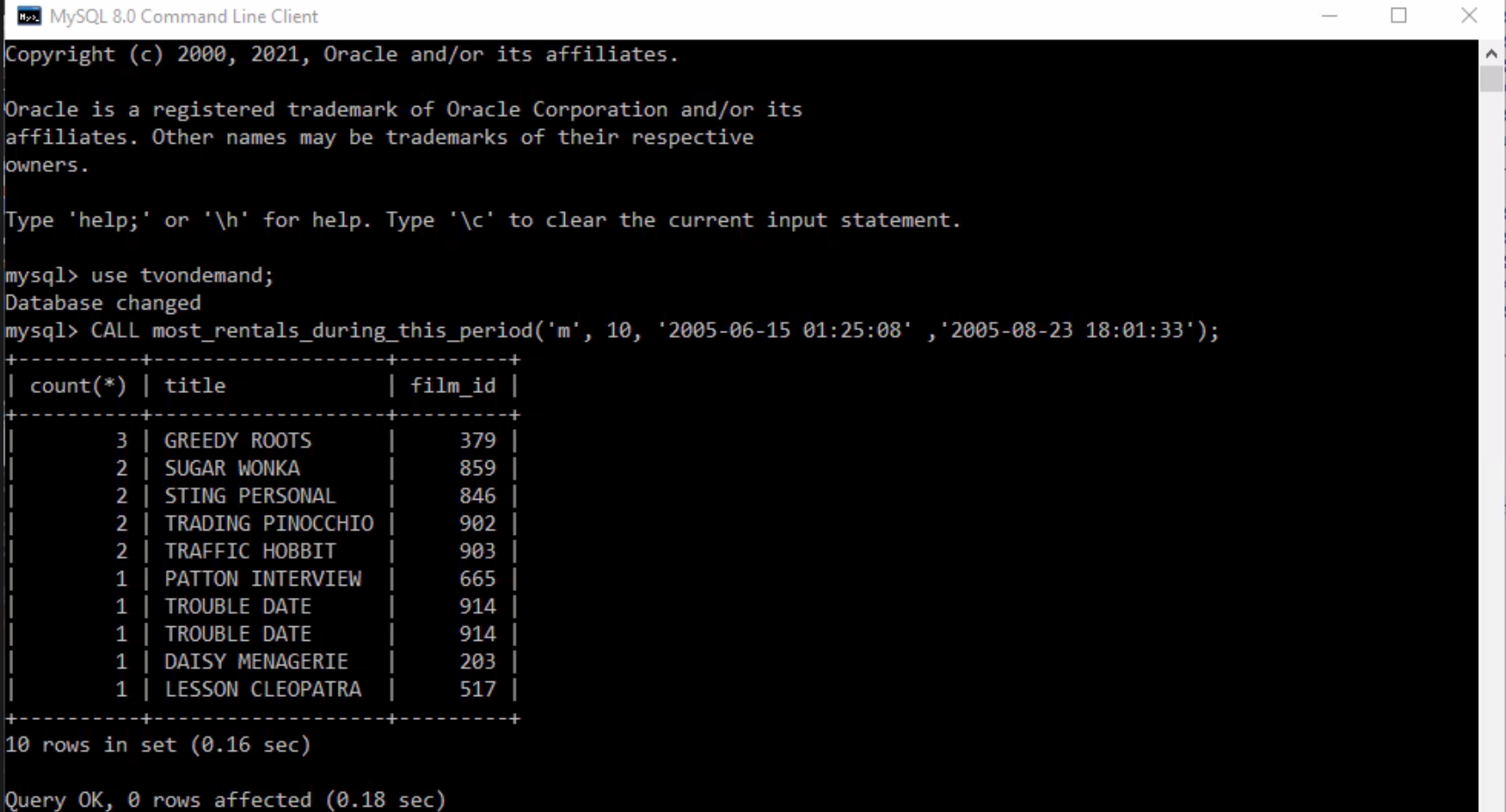
END IF;

END$

DELIMITER ;

CALL most\_rentals\_during\_this\_period('m', 10, '2005-06-15 01:25:08' ,'2005-08-23 18:01:33');

και το αποτέλεσμα της εκτέλεσης του κώδικα είναι το εξής:



3.2

DELIMITER $

DROP PROCEDURE IF EXISTS rentals\_of\_date$

CREATE PROCEDURE rentals\_of\_date(IN cust\_email VARCHAR(50), IN date1 DATETIME, OUT sum\_of\_rentals INT(10))

BEGIN

DECLARE end\_date DATETIME;

DECLARE start\_date DATETIME;

SET @start\_date = date1;

SET @end\_date = DATE\_ADD(date1, INTERVAL 1 DAY); /\*end\_date = date1 + 1 day \*/

SELECT @start\_date;

SELECT @end\_date;

SELECT count(\*) INTO sum\_of\_rentals

FROM customer

INNER JOIN rental ON customer.customer\_id=rental.customer\_id

WHERE customer.email=cust\_email AND rental.rental\_date >= @start\_date AND rental.rental\_date < @end\_date

GROUP BY rental.customer\_id;

IF (@sum\_of\_rentals = null) THEN

set @sum\_of\_rentals = 0;

end if;

END$

DELIMITER ;

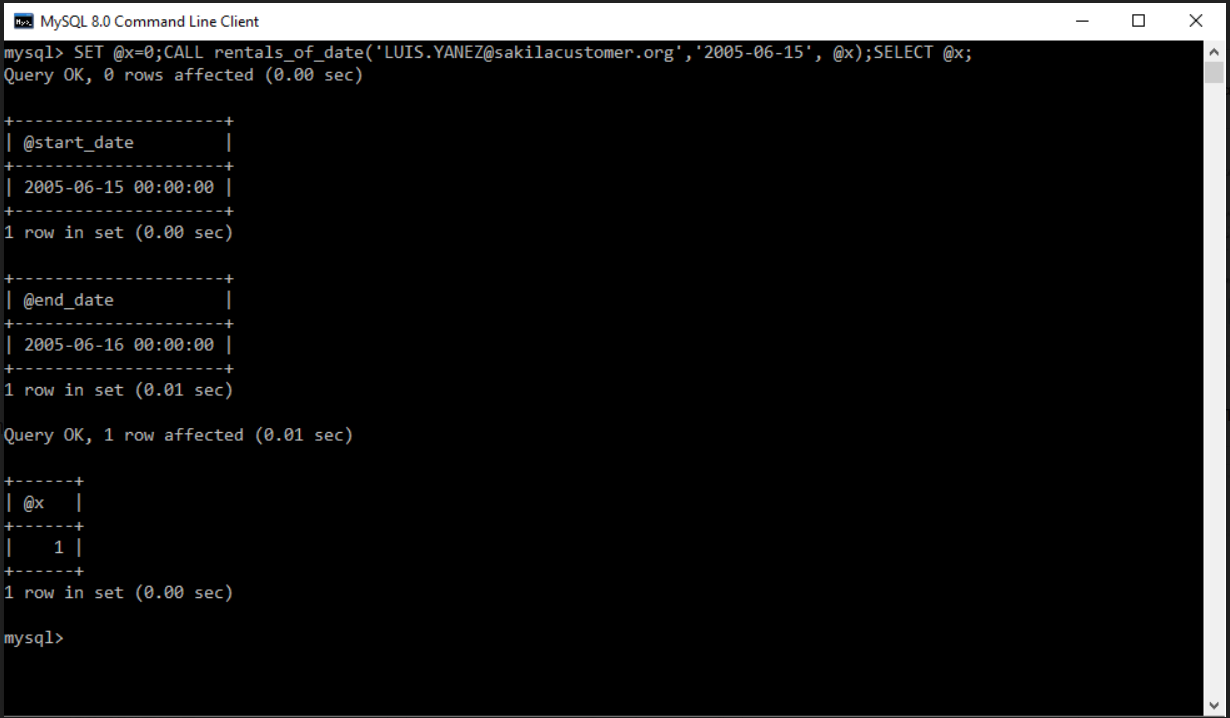
/\*------------------------------------------------------------------------\*/

SET @x=0;

CALL rentals\_of\_date('LUIS.YANEZ@sakilacustomer.org','2005-06-15', @x);

SELECT @x;

και το αποτέλεσμα του κώδικα είναι το εξής:



3.3

use tvondemand;

/\* 3η stored procedure \*/

DELIMITER $

DROP PROCEDURE IF EXISTS esoda\_from\_payments\_per\_month$

CREATE PROCEDURE esoda\_from\_payments\_per\_month ()

BEGIN

DECLARE series\_income DECIMAL(5,2);

DECLARE movies\_income DECIMAL(5,2);

DECLARE end\_date DATETIME;

DECLARE start\_date DATETIME;

DECLARE final\_date DATETIME;

SET series\_income = 0.0;

SET movies\_income =0.0;

SET @start\_date = '2005-05-01 00:00:00';

SET @end\_date = DATE\_ADD(@start\_date, INTERVAL 1 MONTH);

/\*select @start\_date;\*/

/\*select @end\_date;\*/

SELECT MAX(rental\_date) INTO final\_date

FROM rental;

/\*select final\_date;\*/

WHILE (@end\_date<=DATE\_ADD(final\_date, INTERVAL 1 MONTH)) DO

SET series\_income = 0.0;

SET movies\_income =0.0;

SELECT SUM(amount) INTO movies\_income

FROM payment

INNER JOIN rental ON payment.rental\_id = rental.rental\_id

INNER JOIN inventory ON inventory.inventory\_id = rental.inventory\_id

where payment.payment\_date >= @start\_date AND payment.payment\_date < @end\_date AND inventory.series\_id IS null;

IF (movies\_income IS NULL) THEN

SELECT @start\_date, @end\_date, 'NO MOVIES INCOME';

ELSE

SELECT @start\_date, @end\_date ,movies\_income, 'MOVIES';

END IF;

SELECT SUM(amount) INTO series\_income

FROM payment

INNER JOIN rental ON payment.rental\_id = rental.rental\_id

INNER JOIN inventory ON inventory.inventory\_id = rental.inventory\_id

where payment.payment\_date >= @start\_date AND payment.payment\_date < @end\_date AND inventory.film\_id IS null;

IF (series\_income IS NULL) THEN

SELECT @start\_date, @end\_date, 'NO SERIES INCOME';

ELSE

SELECT @start\_date, @end\_date ,movies\_income, 'SERIES';

END IF;

SET @start\_date = DATE\_ADD(@start\_date, INTERVAL 1 MONTH);

SET @end\_date = DATE\_ADD(@end\_date, INTERVAL 1 MONTH);

END WHILE;

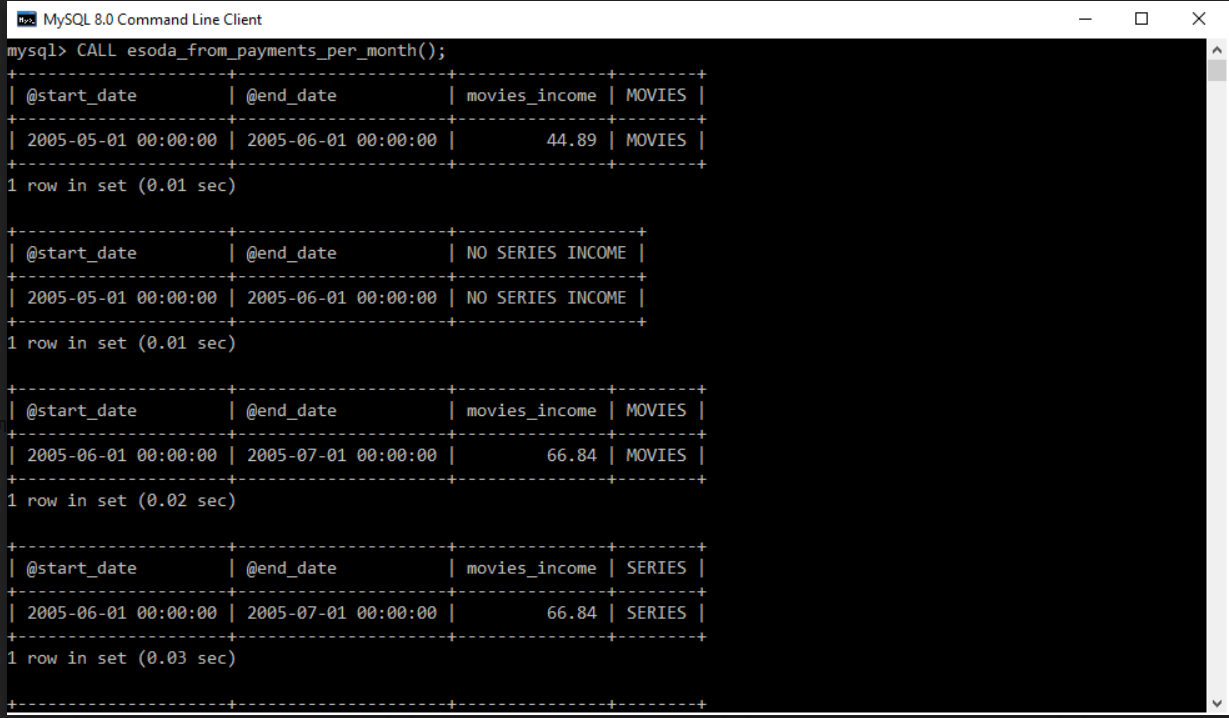
END $

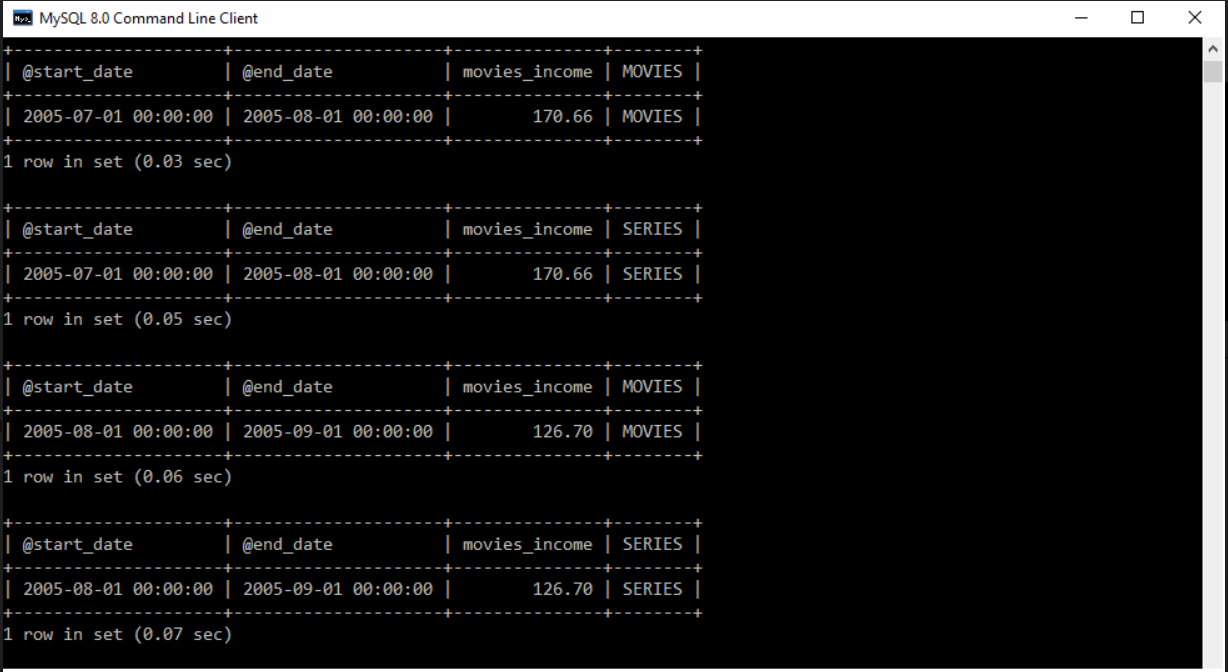
DELIMITER ;

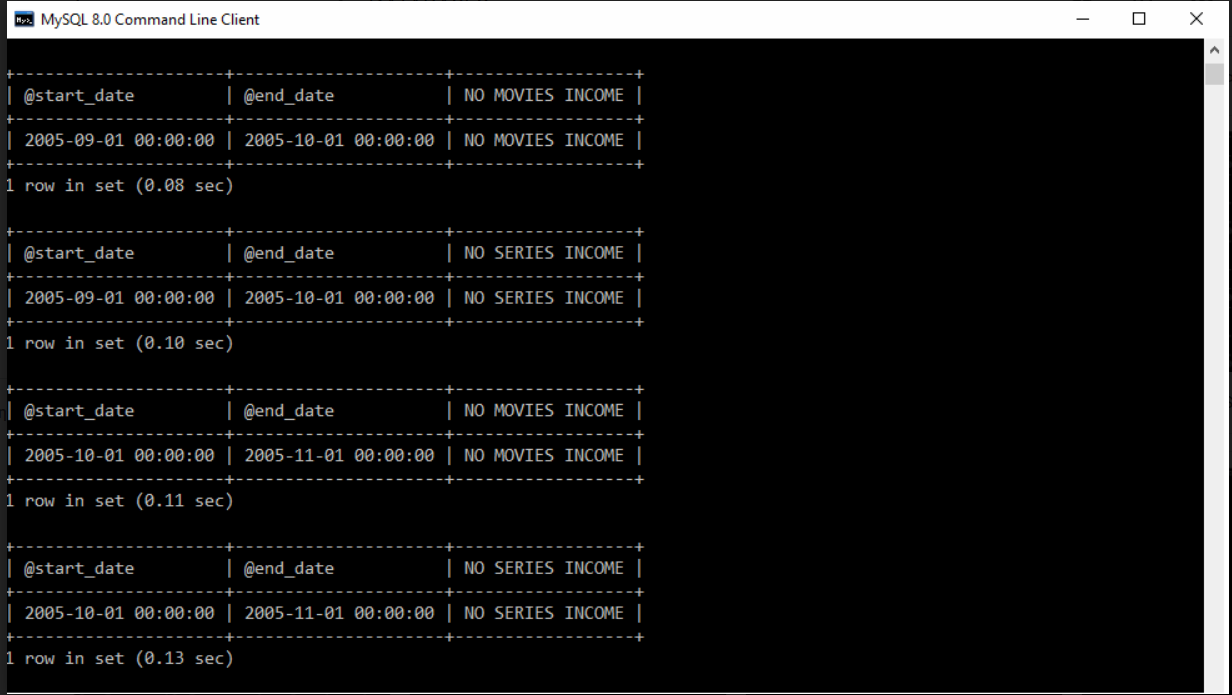
/\* select \* from payment ORDER BY payment\_date ASC; \*/

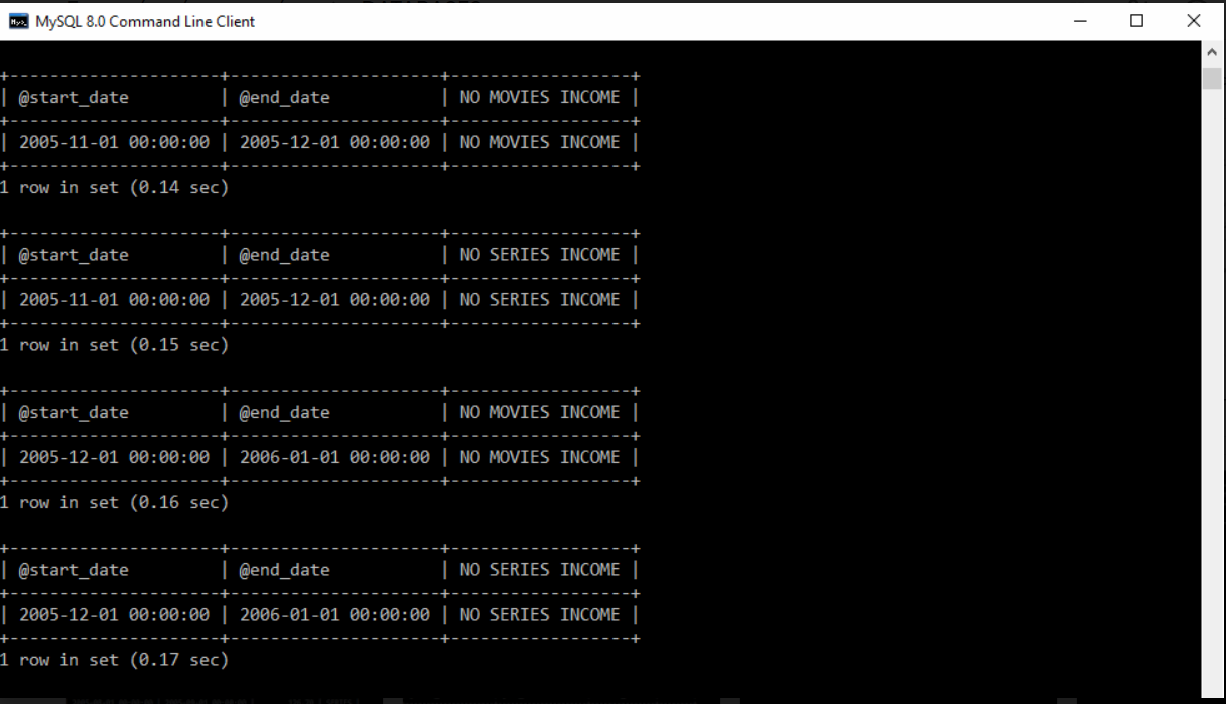
CALL esoda\_from\_payments\_per\_month();

και το αποτέλεσμα της εκτέλεσης του κώδικα είναι το εξής:











3.4α

/\* stored procedure 3.4 α) \*/

/\*DROP INDEX actors\_between\_those\_names\_idx\_1;\*/

/\*DROP INDEX actors\_between\_those\_names\_idx\_2;\*/

CREATE INDEX actors\_between\_those\_names\_idx\_1

ON actor(first\_name);

CREATE INDEX actors\_between\_those\_names\_idx\_2

ON actor(last\_name);

DELIMITER $

DROP PROCEDURE IF EXISTS actors\_between\_those\_names$

CREATE PROCEDURE actors\_between\_those\_names(IN surname1 VARCHAR(45),IN surname2 VARCHAR(45))

BEGIN

/\*DECLARE eponimo1 VARCHAR(45);\*/

/\*DECLARE eponimo2 VARCHAR(45);\*/

/\*SET @eponimo1 = surname1;\*/

/\*SET @eponimo2 = surname2;\*/

SELECT first\_name, last\_name, COUNT(last\_name) AS amount\_of\_actors\_between\_those\_names

FROM actor

/\* WITH (INDEX(actors\_between\_those\_names\_idx\_1)) \*/

WHERE last\_name >= surname1 AND last\_name <= surname2

ORDER BY last\_name ASC;

/\* WHERE last\_name BETWEEN surname1 AND surname2 \*/

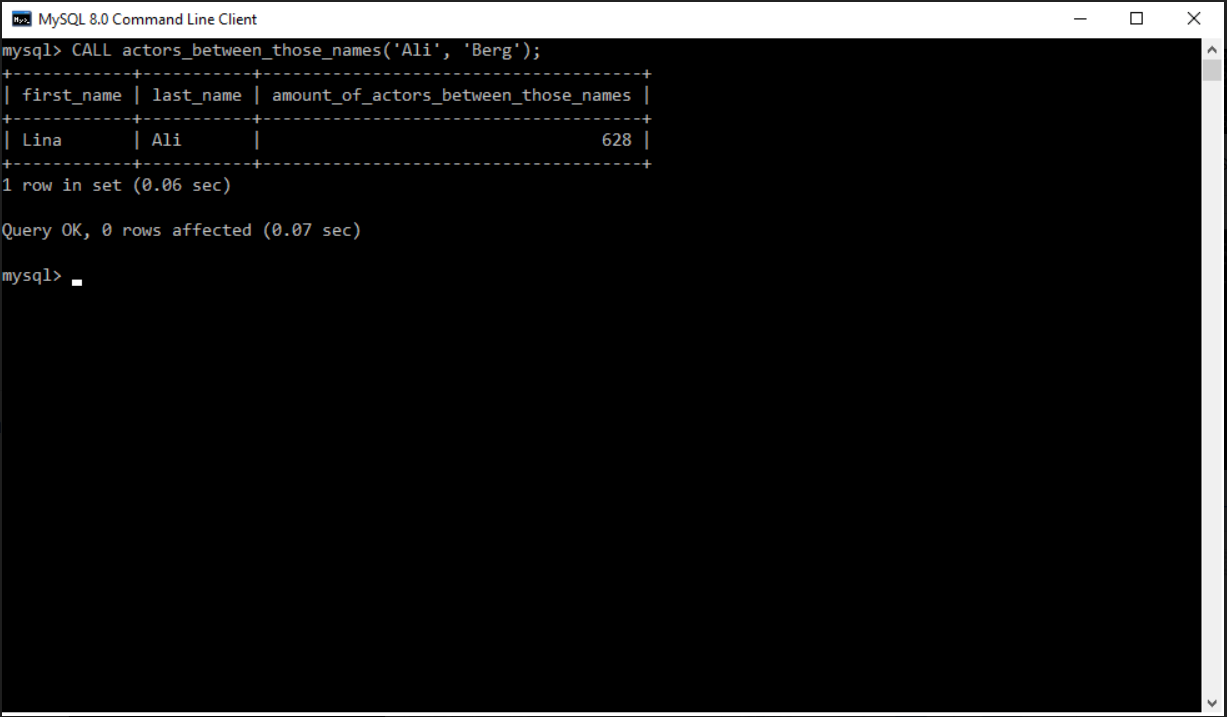
END$

DELIMITER ;

/\*SELECT \* FROM actor ORDER BY last\_name ASC;\*/

CALL actors\_between\_those\_names('Ali', 'Berg');

και το αποτέλεσμα της εκτέλεσης του κώδικα είναι το εξής:



3.4β

/\* stored procedure 3.4 β) \*/

CREATE INDEX actors\_between\_those\_names\_idx\_1

ON actor(first\_name);

CREATE INDEX actors\_between\_those\_names\_idx\_2

ON actor(last\_name);

DELIMITER $

DROP PROCEDURE IF EXISTS actors\_with\_this\_surname$

CREATE PROCEDURE actors\_with\_this\_surname(IN surname VARCHAR(45))

BEGIN

SELECT first\_name, last\_name/\*, COUNT(last\_name)\*/

FROM actor

WHERE last\_name = surname;

IF ( (SELECT COUNT(last\_name) FROM actor WHERE last\_name = surname) >= 2 ) THEN

SELECT COUNT(last\_name) AS amount\_of\_actors\_with\_this\_surname

FROM actor

WHERE last\_name = surname;

END IF;

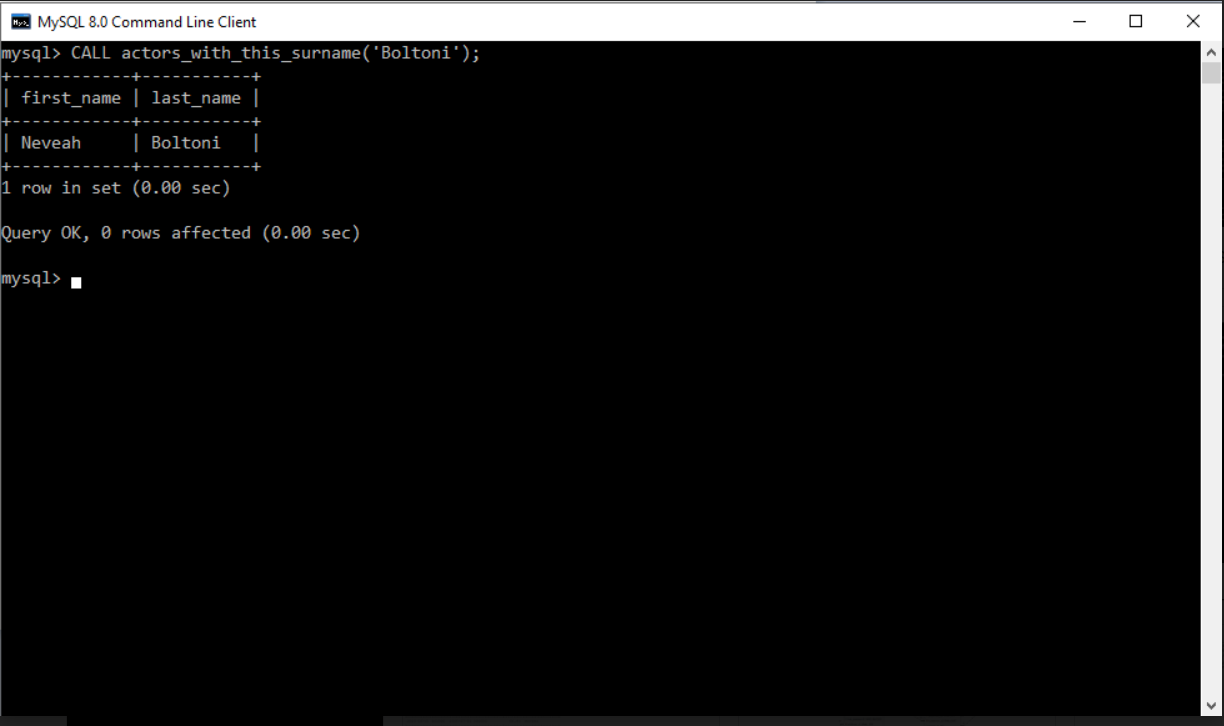
END$

DELIMITER ;

/\*SELECT \* FROM actor ORDER BY last\_name ASC;\*/

CALL actors\_with\_this\_surname('Boltoni'); /\*το επωνυμο Boltoni εχει μονο μια εγγραφη στον πινακα actor, αρα ειναι η οριακη περιπτωση που ζηταμε\*/

και το αποτέλεσμα της εκτέλεσης του κώδικα είναι το εξής:



Πέρα από τις stored procedures που ζητήθηκαν από την εκφώνηση, χρειάστηκε επιπρόσθετα να υλοποιήσουμε και τις παρακάτω 3 stored procedures οι οποίες χρησιμεύουν για τον έλεγχο του login των customer, employee και administrator:

1. stored procedure για ελεγχο login χρηστη customer

DELIMITER $

DROP PROCEDURE IF EXISTS LoginCheckForCustomer$

CREATE PROCEDURE LoginCheckForCustomer (IN email\_given\_from\_user VARCHAR(50), IN password\_given\_from\_user VARCHAR(50), OUT login TINYINT(1) )

BEGIN

DECLARE email\_given VARCHAR(50);

DECLARE password\_given VARCHAR(50);

SET email\_given = email\_given\_from\_user;

SET password\_given = password\_given\_from\_user;

SELECT first\_name , last\_name

FROM customer

WHERE email = email\_given AND cus\_password = password\_given;

IF ( (SELECT COUNT(first\_name) FROM customer WHERE email = email\_given AND cus\_password = password\_given) = 1 AND

(SELECT COUNT(last\_name) FROM customer WHERE email = email\_given AND cus\_password = password\_given) = 1

) THEN

SET login=1;

ELSE

SET login=0;

END IF;

END$

DELIMITER ;

SET @x=0;

CALL LoginCheckForCustomer ('ANNA.HILL@sakilacustomer.org' , '!12345!', @x);

SELECT @x;

/\* SELECT \* FROM customer; \*/

***------------- ΑΙΤΙΟΛΟΓΗΣΗ ΤΟΥ ΚΩΔΙΚΑ -----------------***

Δίνονται ως ορίσματα εισόδου IN το email και ο κωδικός του χρήστη και μια μεταβλητή OUT

Στην IF έπρεπε να βρούμε μια λογική συνθήκη ώστε όταν ο χρήστης δίνει τα σωστά στοιχεία να εκτυπώνει welcome.

Αν βάζαμε μέσα στη συνθήκη της if όλη τη SELECT first\_name , last\_name

FROM customer

WHERE email = email\_given AND cus\_password = password\_given;

θα βγάλει λογικό σφάλμα διότι το αποτέλεσμα της συγκεκριμένης select (εφόσον τα δεδομένα που δίνει η χρήστης είναι σωστά και περιέχονται στη ΒΔ) είναι 2 στήλες first\_name , last\_name με δεδομένα KAI OXI boolean λογική τιμή (δηλαδή true ή false).

Η if μέσα στη συνθήκη της απαιτεί να υπάρχει μια πρόταση/έκφραση η οποία θα είναι true ή false, πχ x > 10.

Επομένως, για να μετατρέψουμε το αποτέλεσμα που επιστρέφει η select (δηλαδή τις 2 στήλες first\_name , last\_name με δεδομένα) σε λογική τιμή σκεφτήκαμε με την εξής λογική:

Αν ο χρήστης δώσει σωστά δεδομένα , τότε το αποτέλεσμα της SELECT first\_name , last\_name

FROM customer

WHERE email = email\_given AND cus\_password = password\_given;

θα είναι δύο στήλες first\_name , last\_name , δηλαδή 1 και 1 (σε πλήθος).

Άρα, αυτό που χρειάζεται είναι να προσθέσουμε την συνάρτηση COUNT για να μετρήσει αυτό το ‘’ 1 και 1 (σε πλήθος)’’ , διότι αν τα στοιχεία που δώσει είναι λανθασμένα τότε δε θα επιστραφούν οι στήλες first\_name , last\_name , δηλαδή οι δύο στήλες θα είναι 0 και 0 (σε πλήθος).

Άρα, για να είναι το COUNT(first\_name) = 1 ΚΑΙ το COUNT(last\_name) = 1 ( δηλαδή 1 και 1 (σε πλήθος) ) , θα πρέπει οπωσδήποτε ο χρήστης να έχει δώσει σωστά στοιχεία email και password στην αρχή.

Σε οποιαδήποτε άλλη περίπτωση (δηλαδή αν δώσει μόνο σωστό email ή μόνο σωστό password ή κανένα από τα δύο σωστά) τότε το ένα απ’ τα δύο ή και τα δύο COUNT θα έχει τιμή 0, το οποίο παραβαίνει τη συνθήκη εντός της if που υπαγορεύει να είναι ίσα με 1 και τα δύο(λογικό AND) , οπότε εξαιτίας αυτής της παραβίασης το λογικό AND θα είναι false, οπότε η συνθήκη μέσα στην if θα είναι false.

Εντελώς αντίστοιχα είναι και η αιτιολόγηση για τις άλλες δύο stored procedures για τους employee και administrator.

1. stored procedure για ελεγχο login χρηστη employee

DELIMITER $

DROP PROCEDURE IF EXISTS LoginCheckForEmployee $

CREATE PROCEDURE LoginCheckForEmployee ( IN email\_given\_from\_user VARCHAR(50), IN password\_given\_from\_user VARCHAR(50), OUT login TINYINT(1) )

BEGIN

DECLARE email\_given VARCHAR(50);

DECLARE password\_given VARCHAR(50);

SET email\_given = email\_given\_from\_user;

SET password\_given = password\_given\_from\_user;

SELECT emp\_first\_name , emp\_last\_name

FROM employee

WHERE emp\_email = email\_given AND emp\_password = password\_given;

IF ( (SELECT COUNT(emp\_first\_name) FROM employee WHERE emp\_email = email\_given AND emp\_password = password\_given) = 1 AND

(SELECT COUNT(emp\_last\_name) FROM employee WHERE emp\_email = email\_given AND emp\_password = password\_given) = 1

) THEN

SET login=1;

ELSE

SET login=0;

END IF;

END$

DELIMITER ;

SET @x=0;

CALL LoginCheckForEmployee ('nikos1@gmail.com' , 'nikos1', @x);

SELECT @x;

/\* SELECT \* FROM employee; \*/

1. stored procedure για ελεγχο login χρηστη administrator

DELIMITER $

DROP PROCEDURE IF EXISTS LoginCheckForAdministrator $

CREATE PROCEDURE LoginCheckForAdministrator ( IN email\_given\_from\_user VARCHAR(50), IN password\_given\_from\_user VARCHAR(50), OUT login TINYINT(1))

BEGIN

DECLARE email\_given VARCHAR(50);

DECLARE password\_given VARCHAR(50);

SET email\_given = email\_given\_from\_user;

SET password\_given = password\_given\_from\_user;

SELECT admin\_first\_name , admin\_last\_name

FROM administrator

WHERE admin\_email = email\_given AND admin\_password = password\_given;

IF ( (SELECT COUNT(admin\_first\_name) FROM administrator WHERE admin\_email = email\_given AND admin\_password = password\_given) = 1 AND

(SELECT COUNT(admin\_last\_name) FROM administrator WHERE admin\_email = email\_given AND admin\_password = password\_given) = 1

) THEN

SET login=1;

ELSE

SET login=0;

END IF;

END$

DELIMITER ;

SET @x=0;

CALL LoginCheckForAdministrator ('nikos1@gmail.com' , 'nikos1', @x);

SELECT @x;

/\* SELECT \* FROM administrator; \*/

***Κεφάλαιο 4: Triggers***

4.1 και 4.2

ο κώδικας του trigger 4.1 είναι ο εξής:

USE tvondemand;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DROP TRIGGER IF EXISTS payment\_insert;

DELIMITER $

CREATE TRIGGER payment\_insert AFTER INSERT ON payment

FOR EACH ROW

BEGIN

INSERT INTO log\_table VALUES(DEFAULT, NULL, NEW.payment\_id, 'NEW PAYMENT (INSERT)', NEW.customer\_id, CURDATE());

END$

DELIMITER ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DROP TRIGGER IF EXISTS payment\_update;

DELIMITER $

CREATE TRIGGER payment\_update AFTER UPDATE ON payment

FOR EACH ROW

BEGIN

INSERT INTO log\_table VALUES(DEFAULT, NULL, NEW.payment\_id, 'PAYMENT UPDATED (UPDATE)', NEW.customer\_id, CURDATE());

END$

DELIMITER ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DROP TRIGGER IF EXISTS payment\_delete;

DELIMITER $

CREATE TRIGGER payment\_delete BEFORE DELETE ON payment

FOR EACH ROW

BEGIN

DECLARE rental\_id\_var INT;

DECLARE customer\_id\_var SMALLINT UNSIGNED;

SET @rental\_id\_var = OLD.rental\_id;

SET @customer\_id\_var = OLD.customer\_id;

INSERT INTO log\_table VALUES(DEFAULT, NULL, @rental\_id\_var, 'RENTAL DELETED (DELETE)', @customer\_id\_var, CURDATE());

END$

DELIMITER ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DROP TRIGGER IF EXISTS rental\_insert;

DELIMITER $

CREATE TRIGGER rental\_insert AFTER INSERT ON rental

FOR EACH ROW

BEGIN

INSERT INTO log\_table VALUES(DEFAULT, NEW.rental\_id, NULL, 'NEW RENTAL (INSERT)', NEW.customer\_id, CURDATE());

END$

DELIMITER ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DROP TRIGGER IF EXISTS rental\_update;

DELIMITER $

CREATE TRIGGER rental\_update AFTER UPDATE ON rental

FOR EACH ROW

BEGIN

INSERT INTO log\_table VALUES(DEFAULT, NEW.rental\_id, NULL, 'RENTAL UPDATED (UPDATE)', NEW.customer\_id, CURDATE());

END$

DELIMITER ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DROP TRIGGER IF EXISTS rental\_delete;

DELIMITER $

CREATE TRIGGER rental\_delete BEFORE DELETE ON rental

FOR EACH ROW

BEGIN

DECLARE rental\_id\_var INT;

DECLARE customer\_id\_var SMALLINT UNSIGNED;

SET @rental\_id\_var = OLD.rental\_id;

SET @customer\_id\_var = OLD.customer\_id;

INSERT INTO log\_table VALUES(DEFAULT, @rental\_id\_var, NULL, 'RENTAL DELETED (DELETE)', @customer\_id\_var, CURDATE());

END$

DELIMITER ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ο κώδικας του trigger 4.2 είναι ο εξής:

/\*3.2 Stored procedure στην οποία δίνεται το email ενός πελάτη και μια ημερομηνία.

Επιστρέφει τον αριθμό των ενοικιάσεων που έχει κάνει αυτήν την ημερομηνία.

Οι ενοικιάσεις μπορούν να αφορούν ταινίες ή σειρές ή και τα δυο,

ανάλογα με το τύπο εγγραφής που έχει κάνει στην υπηρεσία.\*/

/\*

Trigger ο οποίος για κάθε τρίτη φορά που ο ίδιος

χρήστης θα αιτείται να δει ταινία

ή επεισόδιο σειράς

μέσα στην ιδία ημέρα θα έχει έκπτωση 50%.

(Μπορεί να χρησιμοποιηθεί το ερώτημα 3.2)\*/

DROP TRIGGER IF EXISTS discountTrigger;

DELIMITER $

CREATE TRIGGER discountTrigger AFTER INSERT ON rental

FOR EACH ROW

BEGIN

DECLARE end\_date DATETIME;

DECLARE start\_date DATETIME;

DECLARE tot\_rentals INT;

DECLARE customer\_reg ENUM('MOVIES','SERIES','SERIES AND MOVIES');

SET @start\_date = NEW.rental\_date;/\*Ορισμός αρχικής ημερομηνίας\*/

SET @end\_date = DATE\_ADD(NEW.rental\_date, INTERVAL 1 DAY); /\*Ορισμός τελικής ημερομηνίας (επόμενη μέρα)\*/

SET @tot\_rentals=0;

SELECT type\_of\_reg INTO @customer\_reg /\*Εύρεση τύπου εγγραφής του πελάτη για την αντίστοιχη χρέωση\*/

FROM customer

INNER JOIN rental ON rental.customer\_id=customer.customer\_id

WHERE rental.customer\_id=NEW.customer\_id

GROUP BY rental.customer\_id;

SELECT count(\*) INTO @tot\_rentals /\*Εύρεση συνόλου αγορών μέσα στη μέρα\*/

FROM rental

WHERE customer\_id = NEW.customer\_id AND rental.rental\_date >= @start\_date AND rental.rental\_date < @end\_date;

INSERT INTO payment VALUES(DEFAULT, NEW.customer\_id, NEW.rental\_id, 0.0, CURDATE()); #INSERT INTO `payment` (`payment\_id`, `customer\_id`, `rental\_id`, `amount`, `payment\_date`) VALUES

IF(@customer\_reg = 'MOVIES') THEN

UPDATE payment SET amount=0.4 WHERE payment.rental\_id=NEW.rental\_id;

IF(MOD(@tot\_rentals,3)=0) THEN

UPDATE payment SET amount=0.2 WHERE payment.rental\_id=NEW.rental\_id;

END IF;

ELSEIF(@customer\_reg = 'SERIES') THEN

UPDATE payment SET amount=0.2 WHERE payment.rental\_id=NEW.rental\_id;

IF(MOD(@tot\_rentals,3)=0) THEN

UPDATE payment SET amount=0.1 WHERE payment.rental\_id=NEW.rental\_id;

END IF;

ELSE

UPDATE payment SET amount=0.1 WHERE payment.rental\_id=NEW.rental\_id;

IF(MOD(@tot\_rentals,3)=0) THEN

UPDATE payment SET amount=0.05 WHERE payment.rental\_id=NEW.rental\_id;

END IF;

END IF;

END$

DELIMITER ;

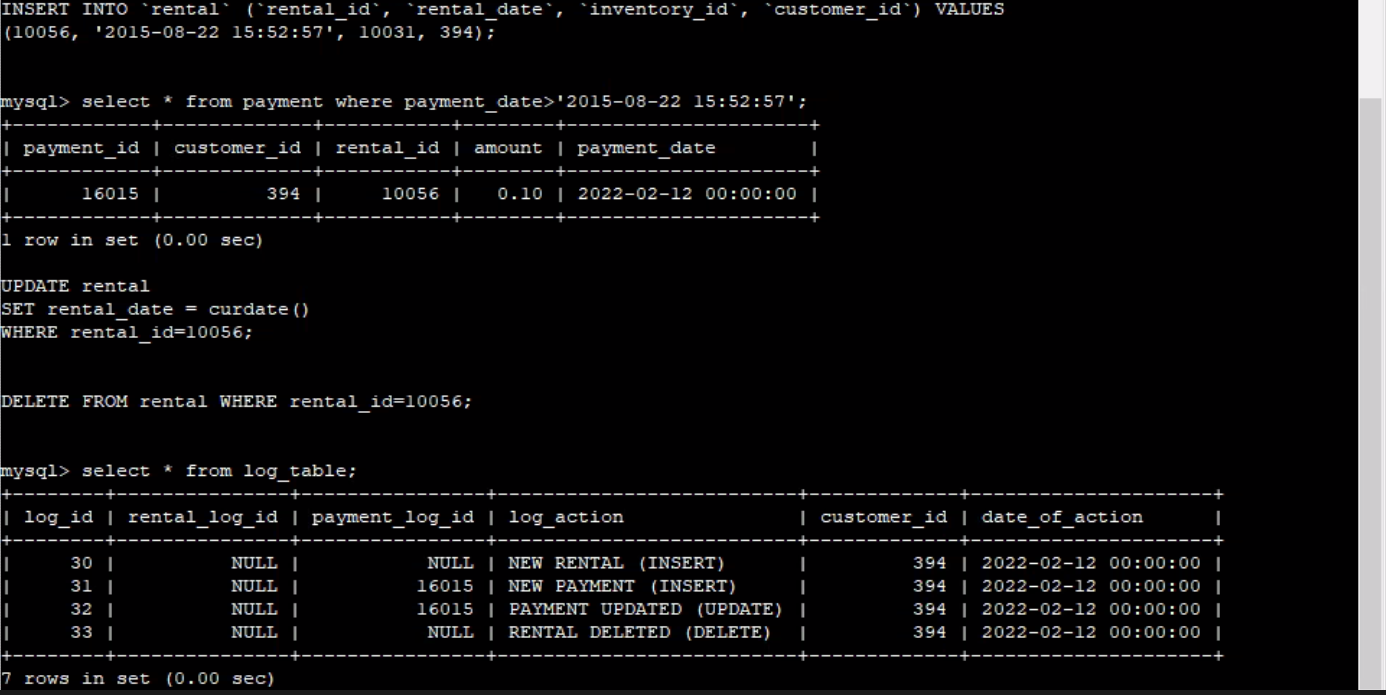
/\*INSERT INTO payment VALUES(DEFAULT, 33, 16000, 0.1, CURDATE());\*/

/\*INSERT INTO rental VALUES(16002, CURDATE(), 3774, 33);\*/

/\*DELETE FROM rental WHERE rental\_id=16002;\*/

/\*(16003, CURDATE(), 3774, 35);\*/

και το αποτέλεσμα των trigger 4.1 (ο 1ος εκτυπωμένος πίνακας της φωτό) και 4.2 (ο 2ος εκτυπωμένος πίνακας της φωτό) είναι το εξής:



Πρώτα κάναμε ένα insert μια νέα εγγραφή ενοικίασης στον πίνακα rental , έπειτα ελέγξαμε την ενοικίαση αυτή ώστε να δούμε αν έχει γίνει η έκπτωση , η οποία έκπτωση δεν έχει γίνει καθώς ο πελάτης αυτός δεν έχει κάνει 3 ενοικιάσεις στην ίδια ημερομηνία, άρα το αποτέλεσμα που βγαίνει είναι σωστό. Έπειτα κάνουμε update στην νέα εγγραφή που κάναμε insert, έπειτα διαγράφουμε την εγγραφή, και τέλος ελέγχουμε τον πίνακα log\_table ώστε να δούμε εάν έχουν καταγραφεί οι ενέργειες.

4.3

ο κώδικας του trigger 4.3 είναι ο εξής:

DROP TRIGGER IF EXISTS ChangeCustomerDataForbidden;

DELIMITER $

CREATE TRIGGER ChangeCustomerDataForbidden

BEFORE UPDATE ON customer

FOR EACH ROW

BEGIN

IF NEW.customer\_id != OLD.customer\_id THEN

SIGNAL SQLSTATE VALUE '45000'

SET MESSAGE\_TEXT = 'Invalid operation! You cannot change your personal customer data.';

END IF;

IF NEW.first\_name != OLD.last\_name THEN

SIGNAL SQLSTATE VALUE '45000'

SET MESSAGE\_TEXT = 'Invalid operation! You cannot change your personal customer data.';

END IF;

IF NEW.last\_name != OLD.last\_name THEN

SIGNAL SQLSTATE VALUE '45000'

SET MESSAGE\_TEXT = 'Invalid operation! You cannot change your personal customer data.';

END IF;

IF NEW.email != OLD.email THEN

SIGNAL SQLSTATE VALUE '45000'

SET MESSAGE\_TEXT = 'Invalid operation! You cannot change your personal customer data.';

END IF;

IF NEW.cus\_password != OLD.cus\_password THEN

SIGNAL SQLSTATE VALUE '45000'

SET MESSAGE\_TEXT = 'Invalid operation! You cannot change your personal customer data.';

END IF;

IF NEW.address\_id != OLD.address\_id THEN

SIGNAL SQLSTATE VALUE '45000'

SET MESSAGE\_TEXT = 'Invalid operation! You cannot change your personal customer data.';

END IF;

IF NEW.active != OLD.active THEN

SIGNAL SQLSTATE VALUE '45000'

SET MESSAGE\_TEXT = 'Invalid operation! You cannot change your personal customer data.';

END IF;

IF NEW.create\_date != OLD.create\_date THEN

SIGNAL SQLSTATE VALUE '45000'

SET MESSAGE\_TEXT = 'Invalid operation! You cannot change your personal customer data.';

END IF;

IF NEW.type\_of\_reg != OLD.type\_of\_reg THEN

SIGNAL SQLSTATE VALUE '45000'

SET MESSAGE\_TEXT = 'Invalid operation! You cannot change your personal customer data.';

END IF;

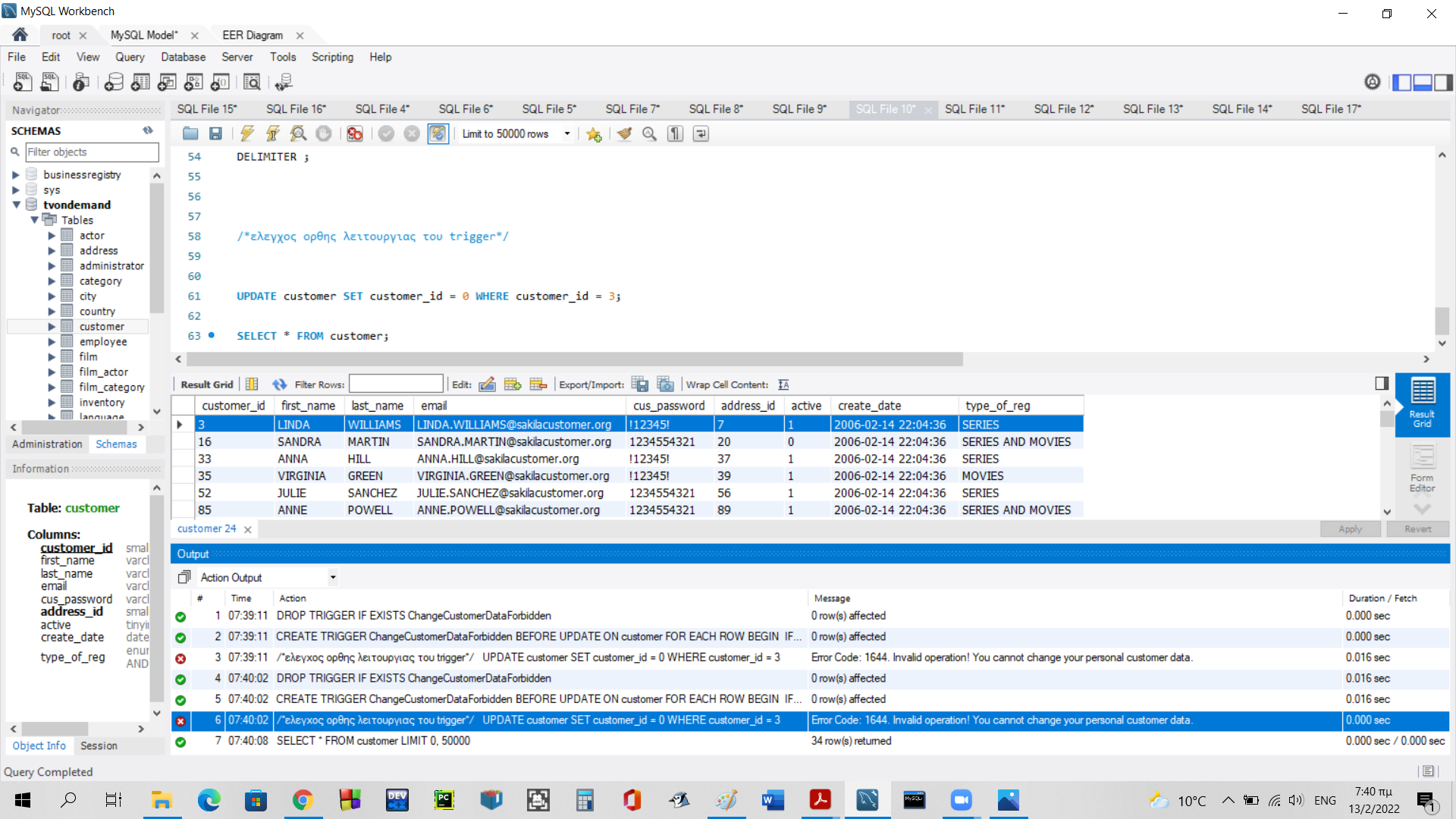
END$

DELIMITER ;

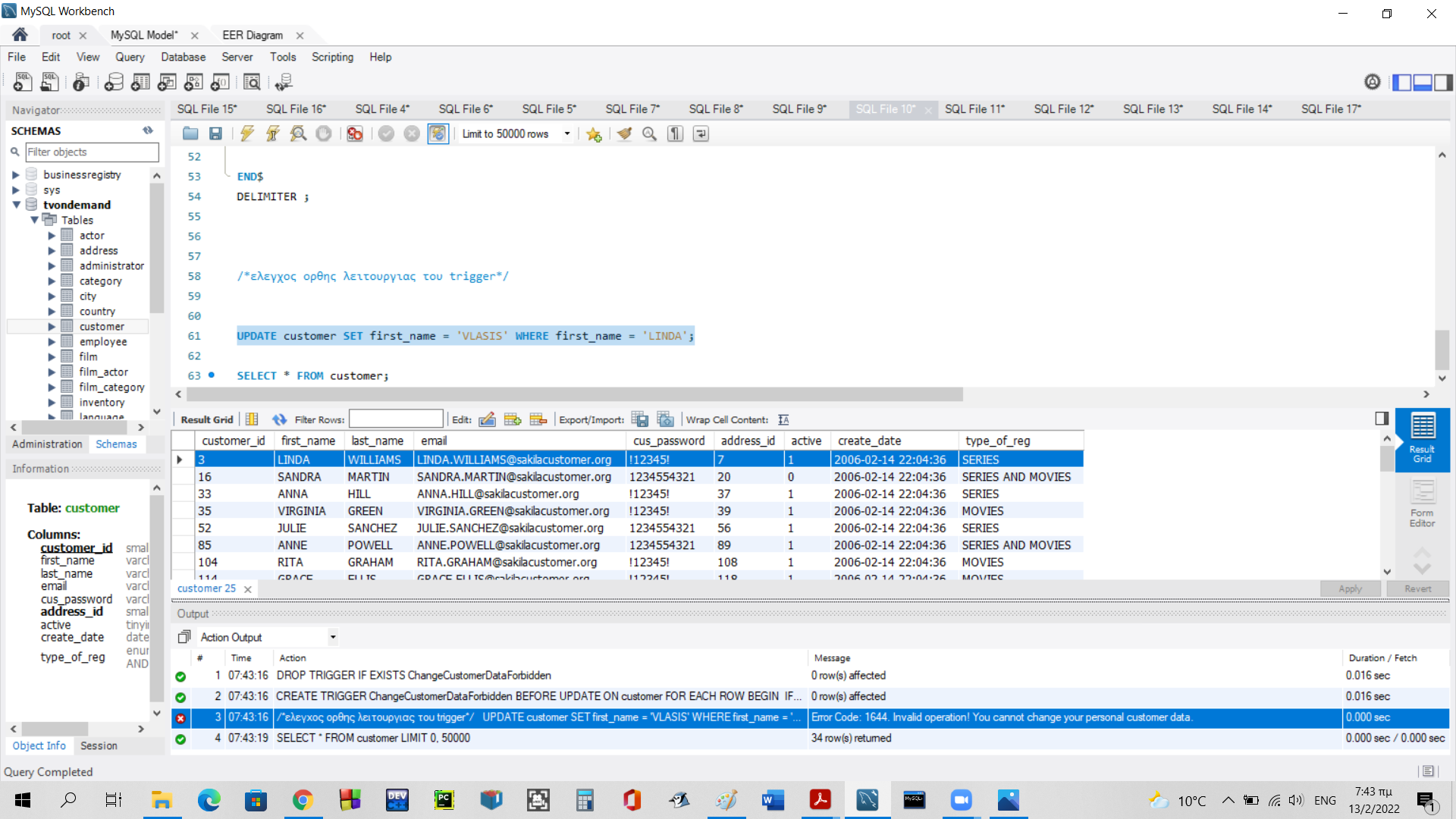
/\*ελεγχος ορθης λειτουργιας του trigger\*/

/\*SELECT \* FROM customer;\*/

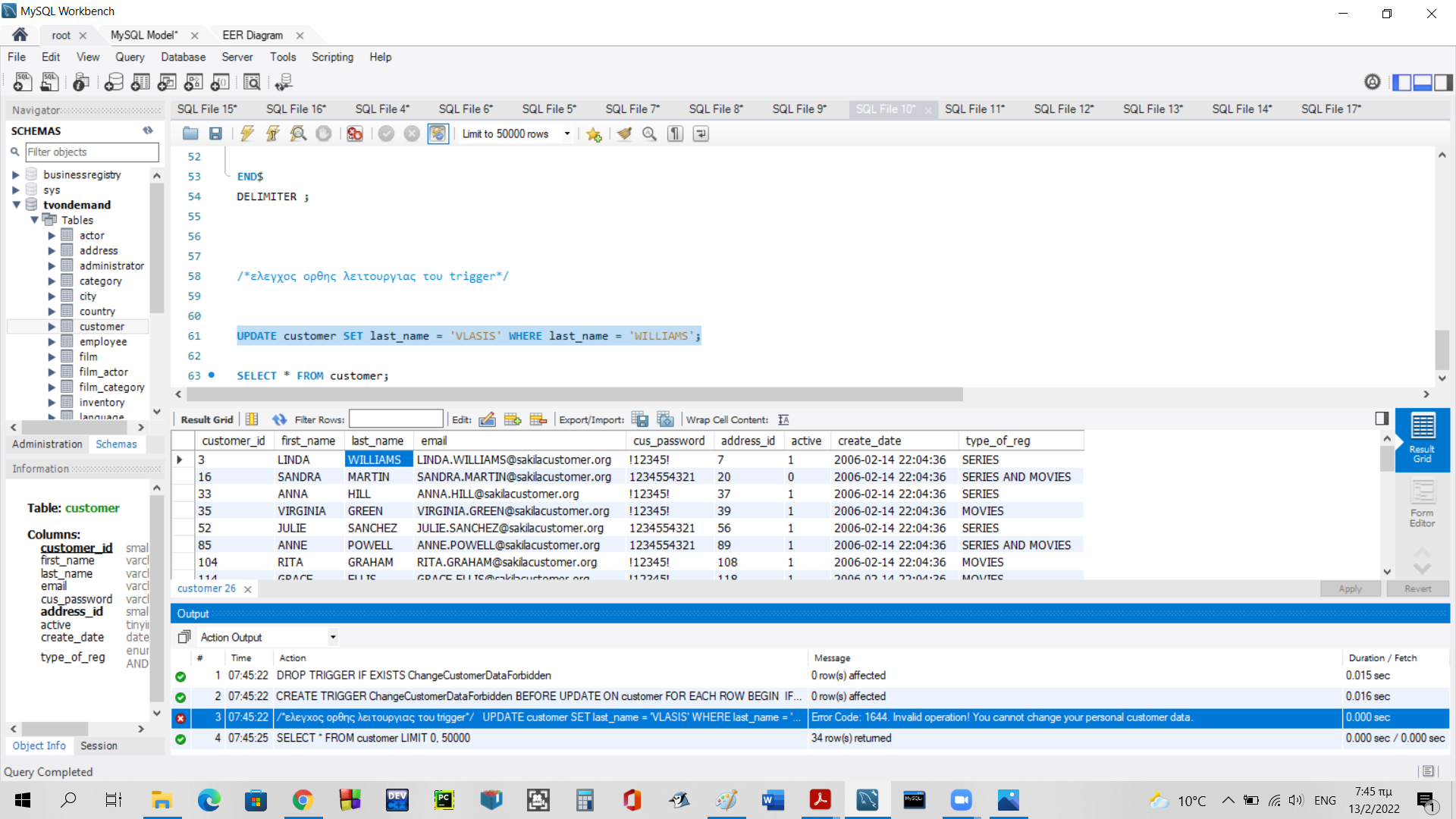
και το αποτέλεσμα του κώδικα για τον έλεγχο της μεταβλητής customer\_id είναι το εξής:



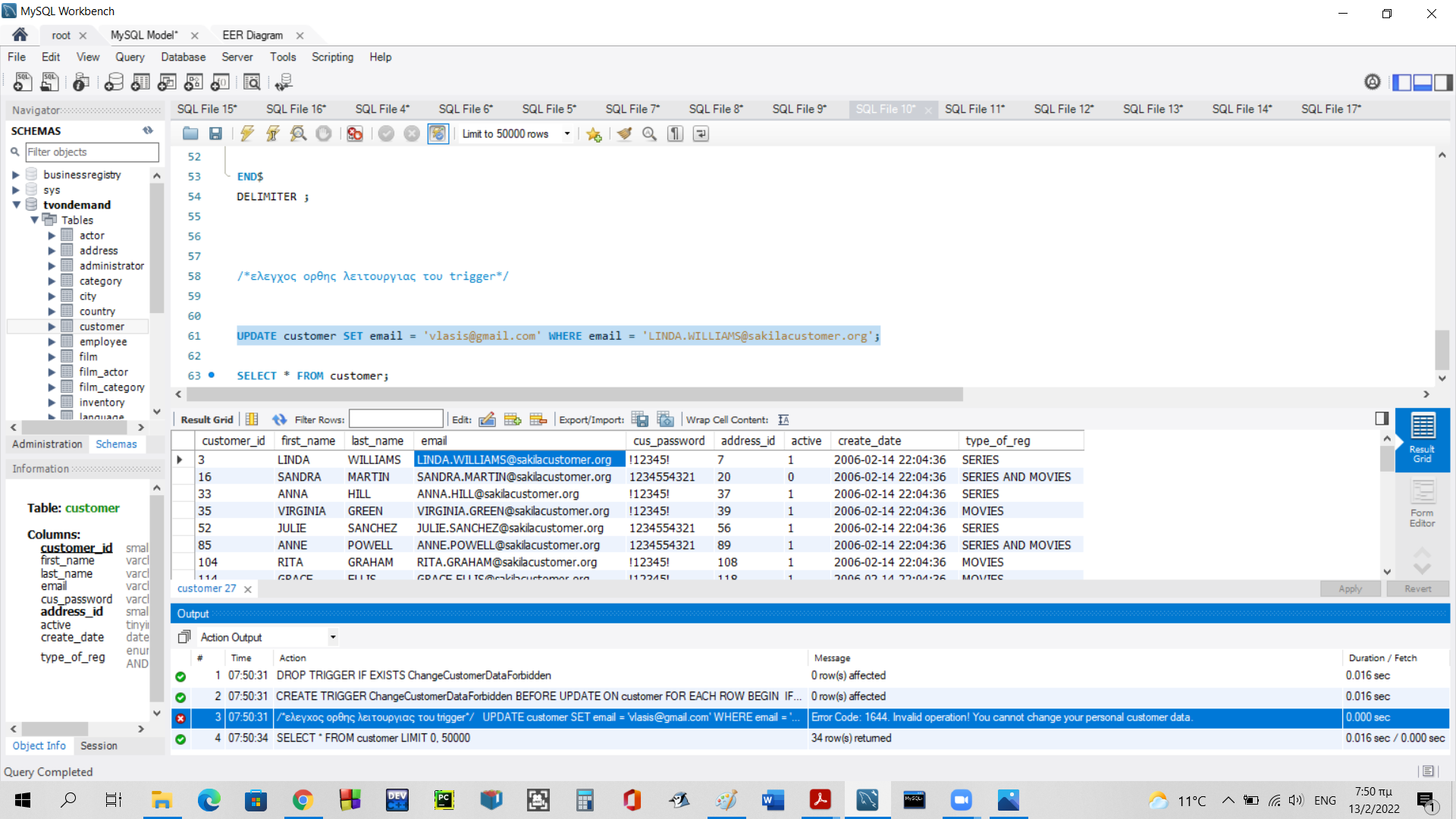
και το αποτέλεσμα του κώδικα για τον έλεγχο της μεταβλητής first\_name είναι το εξής:



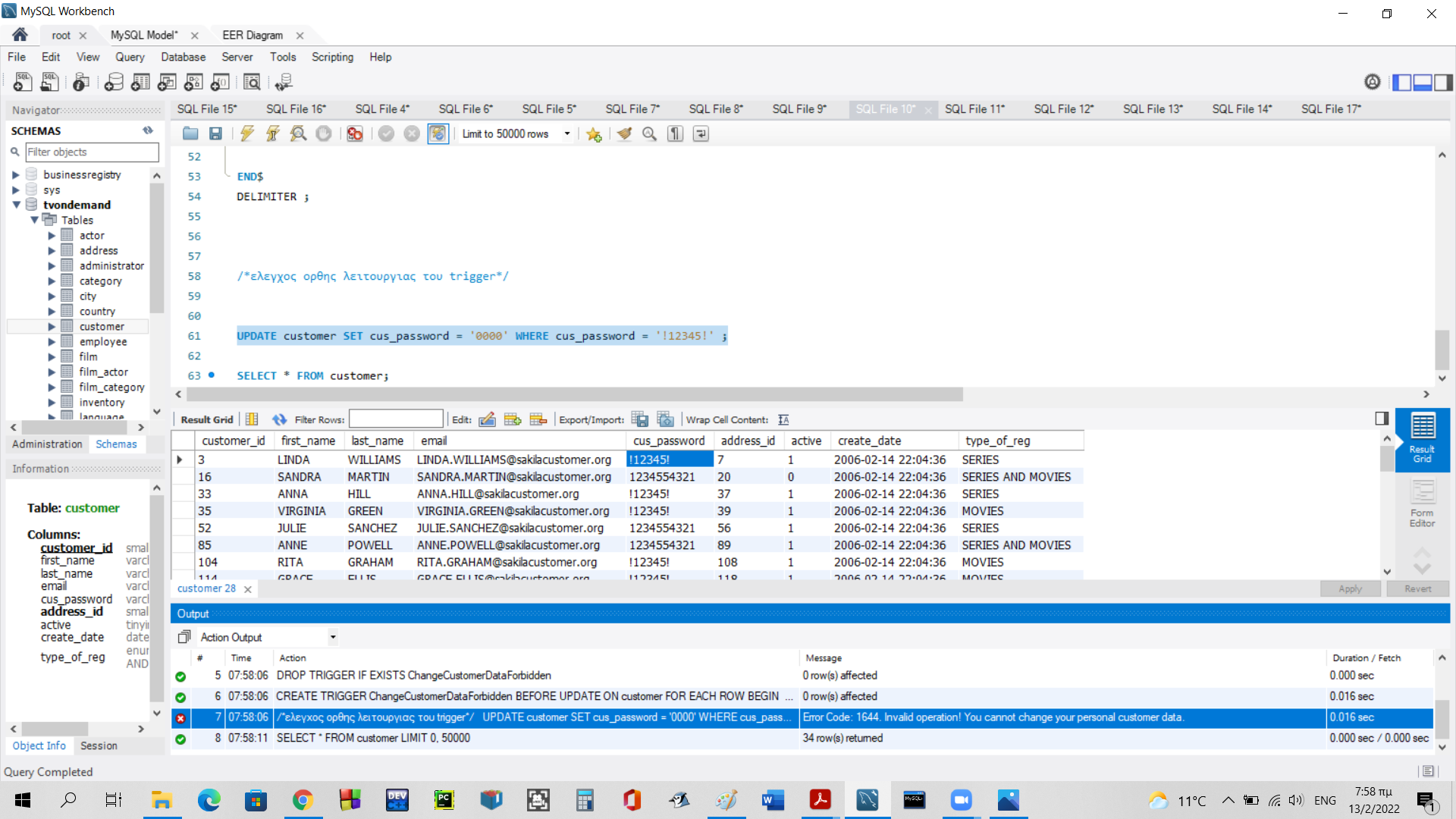
και το αποτέλεσμα του κώδικα για τον έλεγχο της μεταβλητής last\_name είναι το εξής:



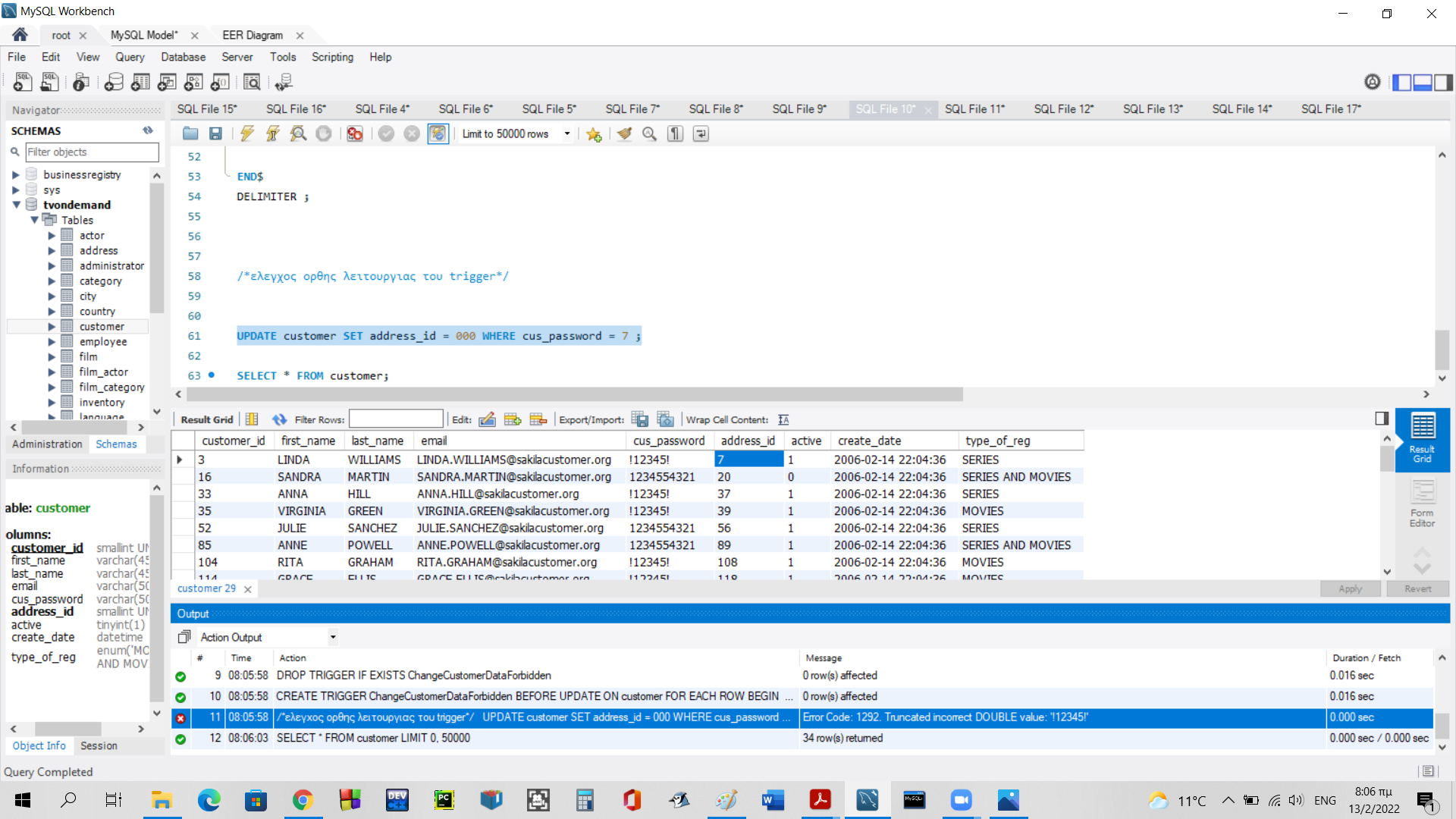
και το αποτέλεσμα του κώδικα για τον έλεγχο της μεταβλητής email είναι το εξής:



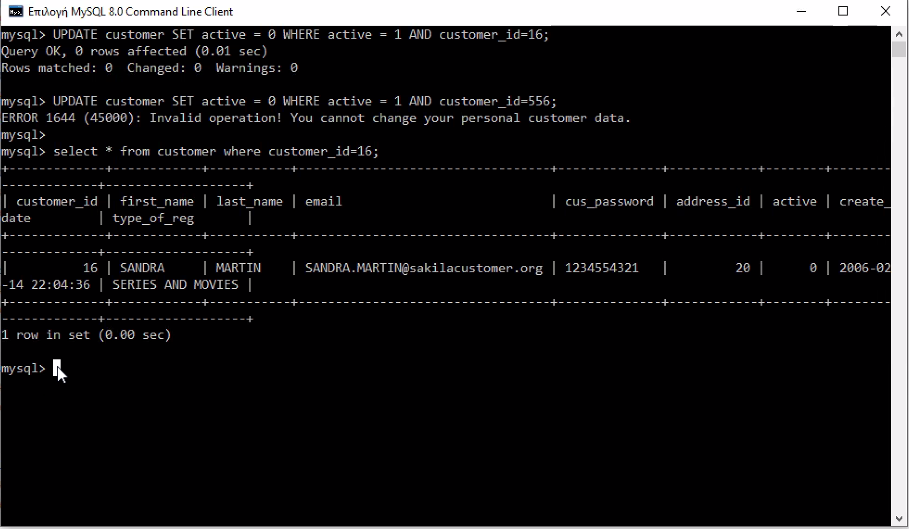
και το αποτέλεσμα του κώδικα για τον έλεγχο της μεταβλητής cus\_password είναι το εξής:

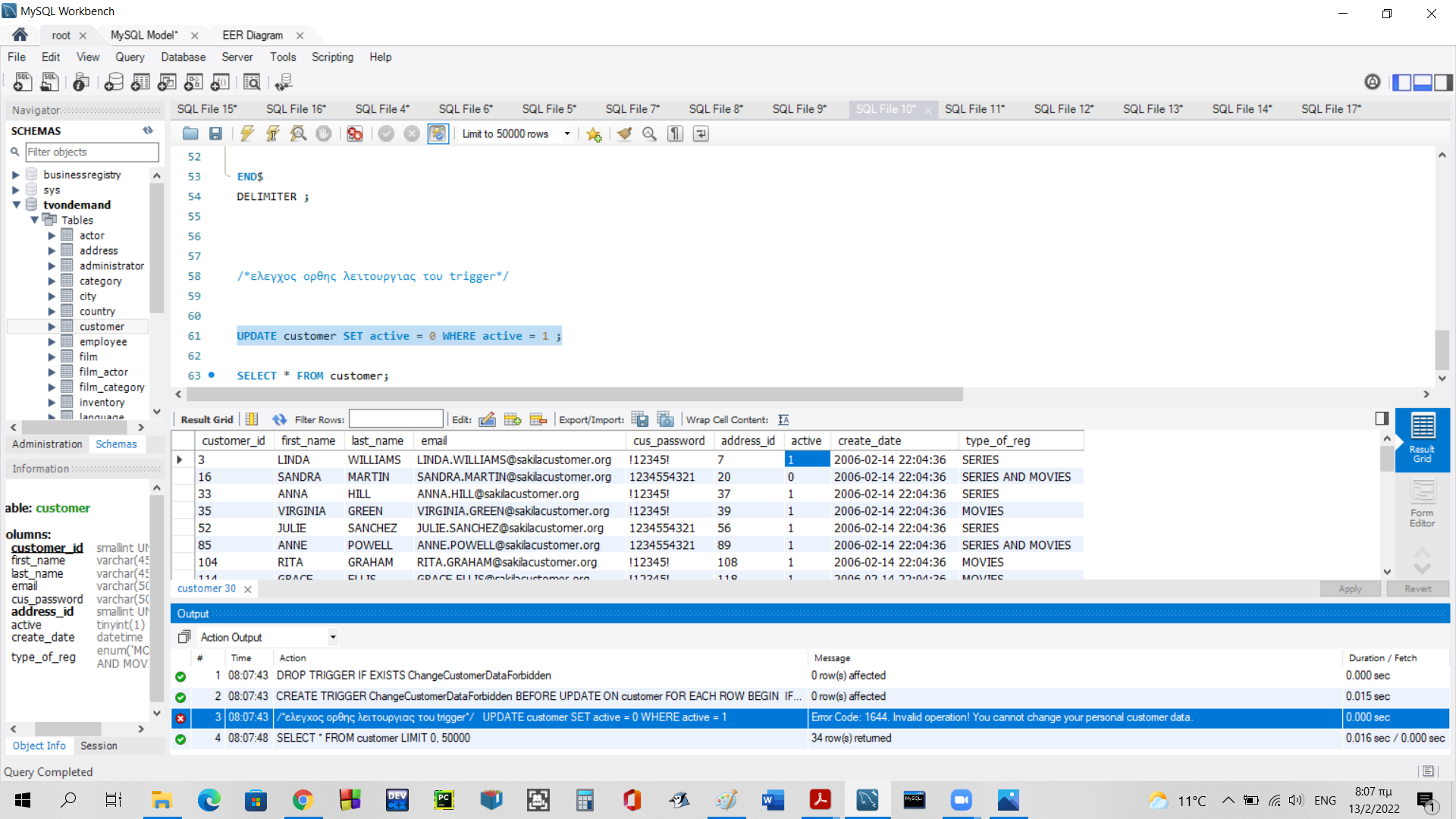


και το αποτέλεσμα του κώδικα για τον έλεγχο της μεταβλητής address\_id είναι το εξής:

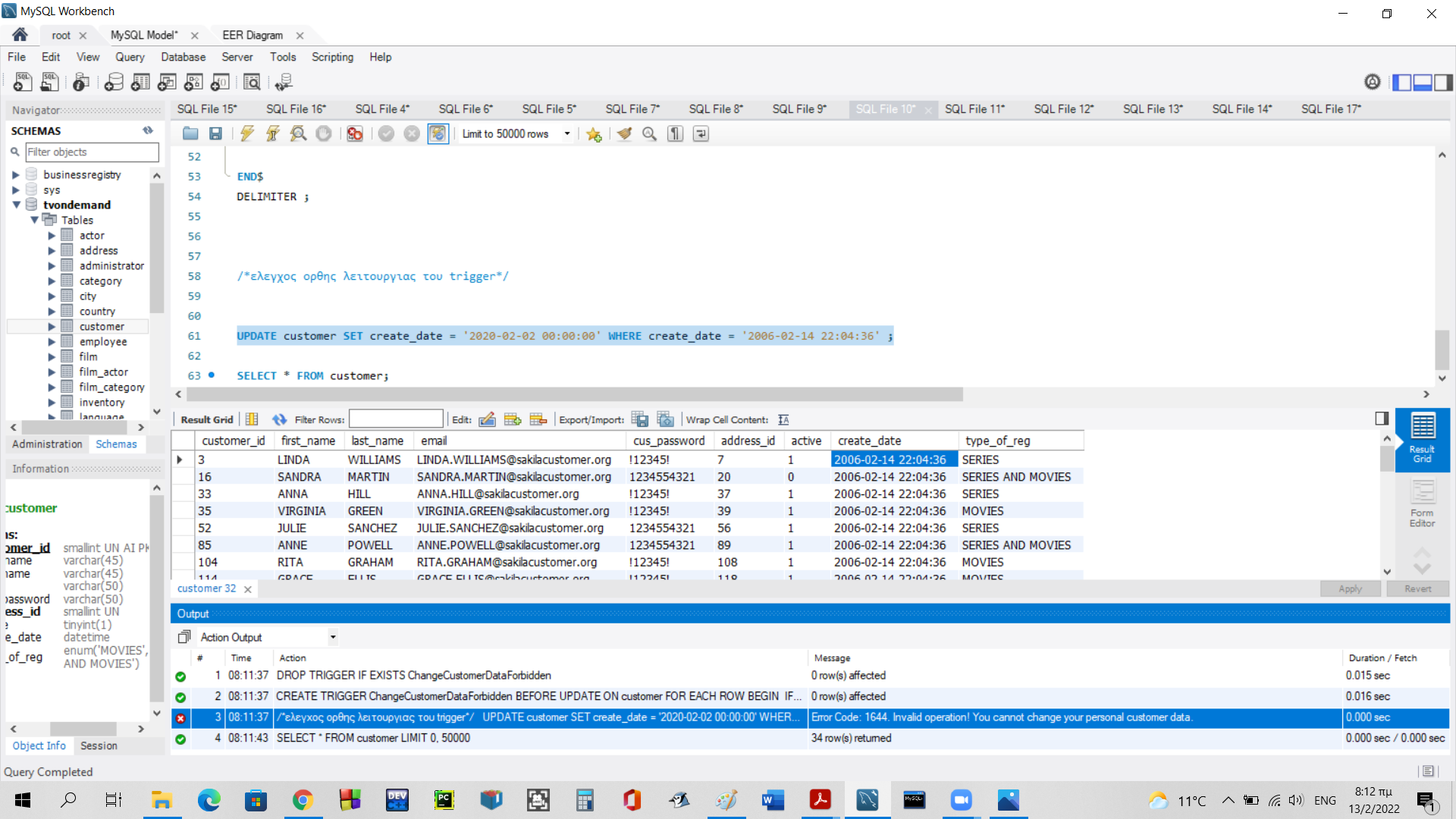


και το αποτέλεσμα του κώδικα για τον έλεγχο της μεταβλητής active είναι το εξής:

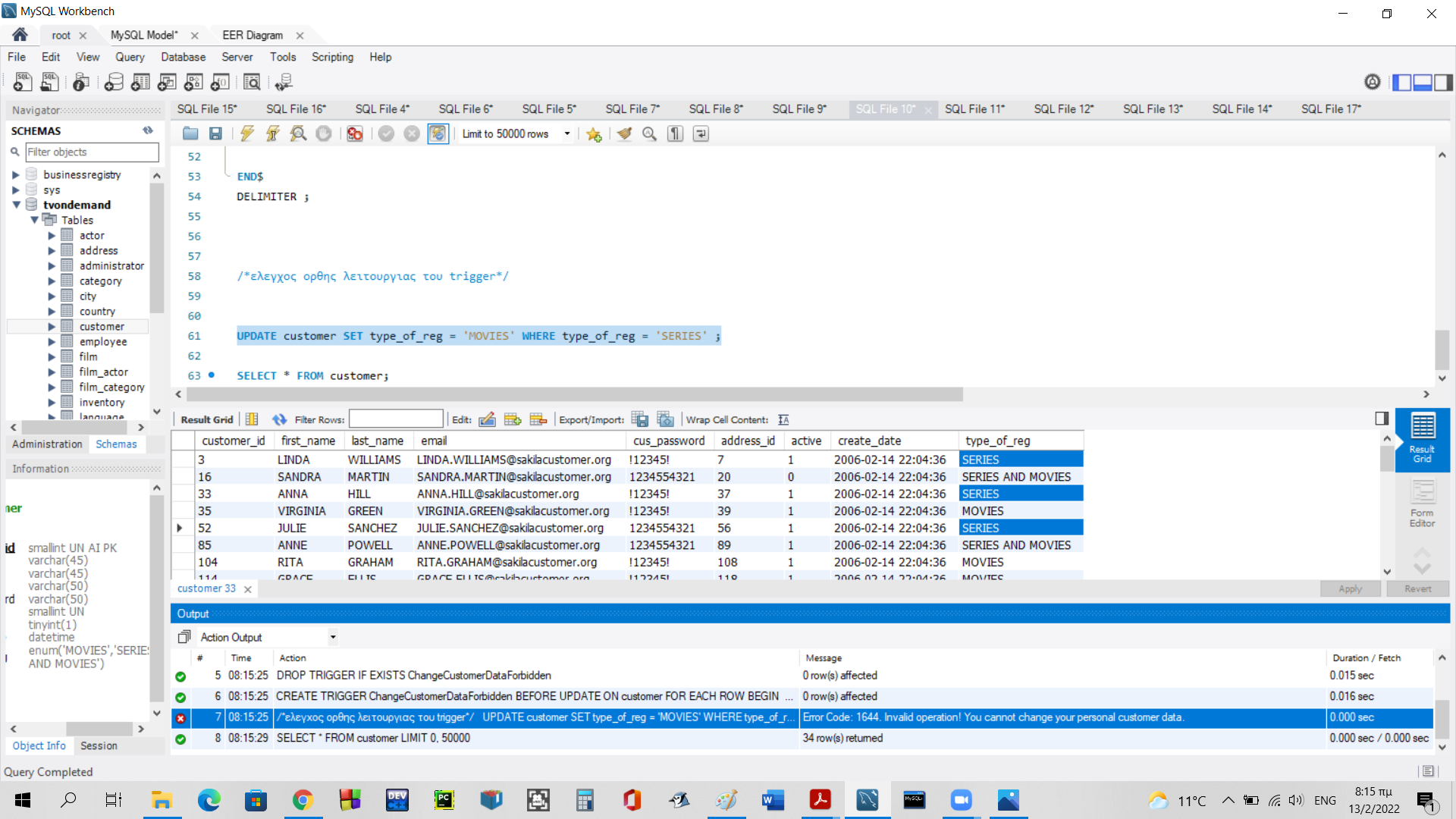




και το αποτέλεσμα του κώδικα για τον έλεγχο της μεταβλητής create\_date είναι το εξής:



και το αποτέλεσμα του κώδικα για τον έλεγχο της μεταβλητής type\_of\_reg είναι το εξής:



***Μέρος Β: GUI***

Στο μέρος Β κατασκευάσαμε το front-end της εφαρμογής, ωστόσο δεν καταφέραμε να το συνδέσουμε με την βάση δεδομένων. Για αυτό το λόγο, θα σας αναφέρουμε περιγραφικά τον τρόπο με τον οποίο θα υλοποιούσαμε κάθε ενέργεια.

Για την λειτουργία της εφαρμογής κάναμε τις παρακάτω παραδοχές:

1. Ο διαχειριστής του συστήματος δε θα μπορεί να δημιουργεί νέους λογαριασμούς για πελάτες,καθώς οι ίδιοι οι πελάτες θα έχουν την δυνατότητα να κάνουν εγγραφή με τα στοιχεία τους στην υπηρεσία (Register) και επίσης δε θα μπορεί να αλλάξει τις τιμές ενοικίασης ταινιών και σειρών καθώς θεωρούμε ότι οι τιμές αυτές είναι σταθερές
2. Οι υπάλληλοι του συστήματος: θα μπορούν να δουν τον φάκελο πελατών αλλά όχι να επεξεργαστούν τα στοιχεία τους
3. Ο υπάλληλος δεν θα μπορεί να ενημερώσει τους πίνακες με πληροφορίες για ηθοποιούς, γλώσσες, κατηγορίες, διευθύνσεις, πόλεις, χώρες, καθώς δεν είχαμε αρκετό χρόνο ώστε να υλοποιήσουμε τα απαραίτητα γραφικά για όλα αυτά. Θα μπορεί ωστόσο να ενημερώνει την βάση προσθέτωντας νέες ταινίες και σειρές.
4. Ο πελάτης θα μπορεί να δει αλλά όχι να επεξεργαστεί τα στοιχεία του λογαριασμού του, συμπεριλαμβανομένου και του τύπου εγγραφής του (εξαιτίας του trigger 4.3 το οποίο δεν επιτρέπει την αλλαγή των στοιχείων ενός λογαριασμού πελάτη).
5. να αλλάξει το τύπο εγγραφής του (μόνο ταινίες, μόνο σειρές ή και τα δύο).

Περιγραφή και τρόπος χρήσης της εφαρμογής (σε αγκύλες {...} οι αντίστοιχες ονομασίες των “JFrameForms”):

Μπαίνοντας στην εφαρμογή ο χρήστης βλέπει μία σελίδα στην οποία καλείται να επιλέξει την αντίστοιχη ιδιότητά του{Login\_WHO} (Admin, Employee, Login), ωστέ να οδηγηθεί στην αντίστοιχη σελίδα ώστε να κάνει Login {AdminLogin, EmployeeLogin, CustomerRegisterLogin} για την περίπτωση του πελάτη θεωρούμε ότι μπορεί να κάνει μόνος του την εγγραφή για τον εαυτό του σε περίπτωση που δεν είναι ήδη εγγεγραμμένος στην υπηρεσία, για αυτό υπάρχει η πρόσθετη σελίδα {CustomerRegisterLogin} στην οποία επιλέγει εάν θέλει να κάνει login ή Register. Στην περίπτωση του login o χρήστης οδηγείται σε μία σελίδα {CustomerLogin} στην οποία θα κάνει την είσοδό του στην υπηρεσία, ωστόσο εάν επιλέξει να κάνει Register θα οδηγηθεί σε μία σελίδα {CustomerRegister}, στην οποία του δίνεται η δυνατότητα να συμπληρώσει ορισμένα στοιχεία ώστε να κάνει εγγραφή στην υπηρεσία, το οποίο θέλοντας να το μεταφράσουμε στην γλώσσα της SQL θα λέγαμε ότι θα κάνει INSERT στον πίνακα customer μία νέα πλειάδα με τα αντίστοιχα στοιχεία του, ενέργεια η οποία θα συνδεόταν με το κουμπί Register της σελίδας.

\*\*String INSERT\_QUERY = "INSERT INTO customer VALUES (...)";

\*\*ResultSet resultSet = statement.executeQuery(INSERT\_QUERY);

\*\*Κάθε τιμή στήλης στο INSERT θα έπαιρνε μία DEFAULT τιμή ή την τιμή \*\*από το αντίστοιχο πεδίο εισαγωγής δεδομένων της σελίδας.

Στην περίπτωση του Login και για τις τρεις οντότητες η διαδικασία είναι ακριβώς η ίδια. Σε μία σελίδα {AdminLogin, EmployeeLogin, CustomerLogin} ο χρήστης εισάγει το email και τον κωδικό του και έπειτα καλείται μία Stored Procedure (αντίστοιχα για Admin, Employee, Customer οι LoginCheckForAdministrator, LoginCheckForEmployee, LoginCheckForCustomer), οι οποία παίρνει ορίσματα (IN email\_given\_from\_user VARCHAR(50), IN password\_given\_from\_user VARCHAR(50), OUT login TINYINT(1) ) δηλαδή ως τα 2 πρώτα ορίσματα το email και τον κωδικό που θα εισάγει ο χρήστης και ως 3ο όρισμα την μεταβλητή στην οποία θα αποθηκευτεί η τιμή 1 σε περίπτωση επιτυχούς Ταυτοποίησης των ορισμάτων με τα αντίστοιχα πεδία στην βάση (Έλεγχος Ταυτοποίησης) και η τιμή 0 σε διαφορετική περίπτωση. Σε περίπτωση επιτυχούς ταυτοποίησης (login=1), ο αντίστοιχος χρήστης οδηγείται στο αντίστοιχο menu {Customermenu, Adminmenu, Employeemenu} (τα μενού παρατίθενται παρακάτω) Στο κάθε μενού υπάρχουν επιλογές ώστε να μπορούν να εκτελεστούν οι ενέργειες που ζητούνται -πλην αυτών για τις οποίες έχουμε κάνει παραδοχές–.

Συγκεκριμένα, παρατίθενται παρακάτω οι λειτουργίες του κάθε “Κουμπιού” στο menu του κάθε Χρήστη.

Customer:

1. Customer Info: Το κουμπί αυτό εκτελεί μία Select του πίνακα Customer όπου ο customer\_id είναι ίσος με τον customer\_id του πελάτη που έκανε Login.
2. Rentals: To κουμπί αυτό κάνει μία Select του πίνακα Rental όπου ο customer\_id είναι ίσος με τον customer\_id του πελάτη που έκανε Login.
3. Show all Movies/Series: Το κουμπί αυτό κάνει μία Select \* από τους πίνακα movies series σε συνδυασμό με τον πίνακα inventory (ώστε να φαίνεται και το ID της κάθε ταινίας ή σειράς) εκτυπώνοντας το σύνολο των ταινιών και/ή των σειρών ανάλογα βέβαια με τον τύπο εγγραφής του εκάστοτε χρήστη.
4. New Rental: Το κουμπί αυτό οδηγεί σε μία νέα σελίδα {newRentalForm} στην οποία ο χρήστης καλείται να συμπληρώσει ορισμένα στοιχεία και να πραγματοποιήσει μέσω του κουμπιού Rent INSERT στον πίνακα Rental με τιμές αντίστοιχες αυτών στα πεδία εισαγωγής δεδομένων
5. Logout: Το κουμπί αυτό πραγματοποιεί έξοδο από την βάση δεδομένων και επιστροφή στην αρχική σελίδα.

Admin:

1. New Employee: Το κουμπί αυτό οδηγεί σε μία νέα σελίδα {newEmployeeForm} στην οποία ο χρήστης καλείται να συμπληρώσει ορισμένα στοιχεία και να πραγματοποιήσει μέσω του κουμπιού Register, INSERT στον πίνακα employee με τιμές αντίστοιχες αυτών στα πεδία εισαγωγής δεδομένων.
2. Delete Customer: Το κουμπί αυτό οδηγεί σε σελίδα όπου ο χρήστης καλείται να εισάγει το όνομα, το επώνυμο και τον ID του customer (για την αποφυγή συνονυμιών) και πατώντας το κουμπί Delete κάνει DELETE από τον πίνακα customer του αντίστοιχου πελάτη
3. Delete Employee: Το κουμπί αυτό οδηγεί σε σελίδα όπου ο χρήστης καλείται να εισάγει το όνομα, το επώνυμο και τον ID του employee (για την αποφυγή συνονυμιών) και πατώντας το κουμπί Delete κάνει DELETE από τον πίνακα employee του αντίστοιχου υπαλλήλου
4. Monthly Incomes: Το κουμπί αυτό κάνει κλήση της Stored Procedure του ερωτήματος 3.3 η οποία εκτυπώνει τα έσοδα ανα μήνα.
5. Logout: Το κουμπί αυτό πραγματοποιεί έξοδο από την βάση δεδομένων και επιστροφή στην αρχική σελίδα.

Employee:

1. Show All Customers: Το κουμπί αυτό εκτελεί SELECT \* από τον πίνακα customer.
2. Show All Rentals: Το κουμπί αυτό εκτελεί SELECT \* από τον πίνακα rental.
3. New Movie: Το κουμπί αυτό οδηγεί σε μία νέα σελίδα {newMovieForm} στην οποία ο χρήστης καλείται να συμπληρώσει ορισμένα στοιχεία και να πραγματοποιήσει μέσω του κουμπιού Insert, INSERT στον πίνακα movies με τιμές αντίστοιχες αυτών στα πεδία εισαγωγής δεδομένων.
4. New Serie: Το κουμπί αυτό οδηγεί σε μία νέα σελίδα {newSerieForm} στην οποία ο χρήστης καλείται να συμπληρώσει ορισμένα στοιχεία και να πραγματοποιήσει μέσω του κουμπιού Insert, INSERT στον πίνακα series, με τιμές αντίστοιχες αυτών στα πεδία εισαγωγής δεδομένων.
5. Show 5 Most Successful Movies: To κουμπί αυτό κάνει κλήση της Stored Procedure του ερωτήματος 3.1 με ορίσματα

call most\_rentals\_during\_this\_period('m', 10, CURDATE() ,DATE\_ADD(CURDATE(), INTERVAL 1 MONTH));

1. Show 5 Most Successful Series: To κουμπί αυτό κάνει κλήση της Stored Procedure του ερωτήματος 3.1 με ορίσματα

call most\_rentals\_during\_this\_period('s', 10, CURDATE() ,DATE\_ADD(CURDATE(), INTERVAL 1 MONTH));

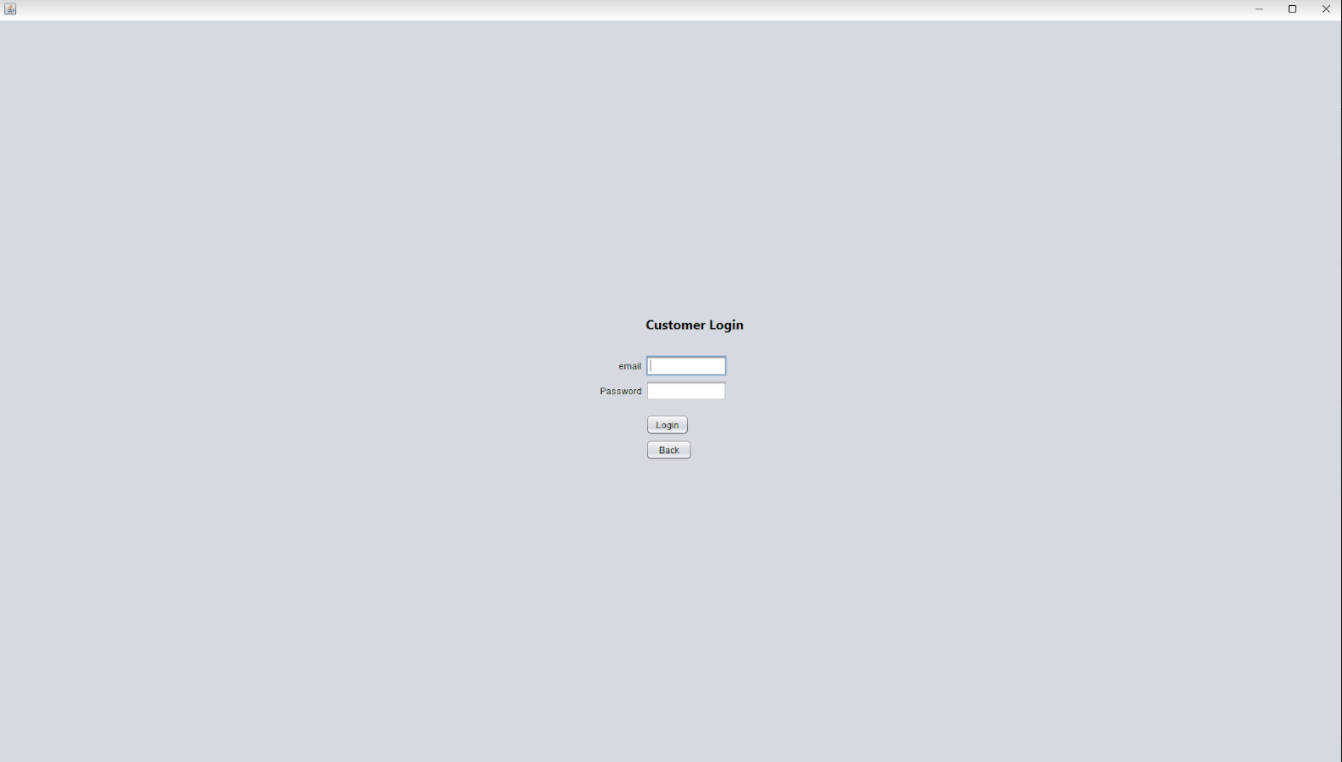
1. Logout: Το κουμπί αυτό πραγματοποιεί έξοδο από την βάση δεδομένων και επιστροφή στην αρχική σελίδα.

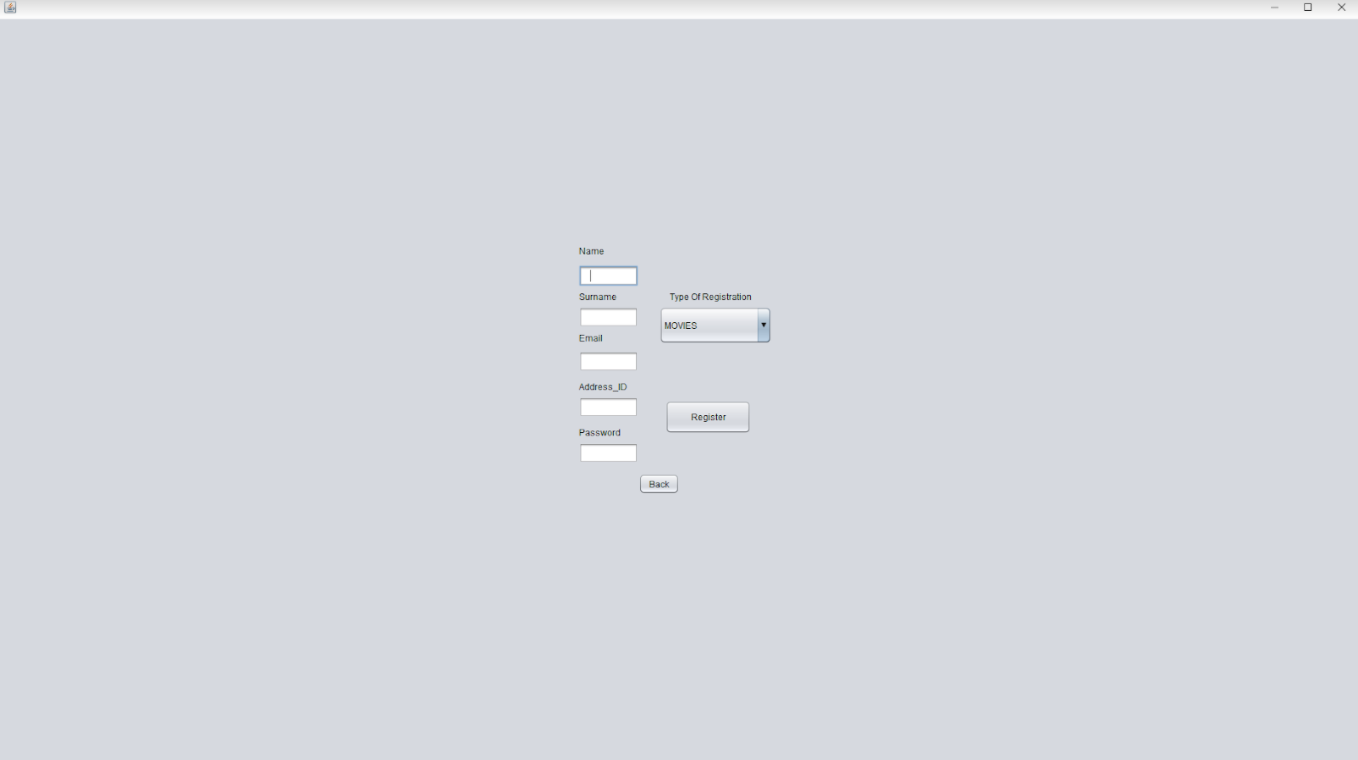
Επίσης στις περισσότερες σελίδες εμφανίζεται το κουμπί Back το οποίο οδηγεί τον χρήστη στην προηγούμενη σελίδα.

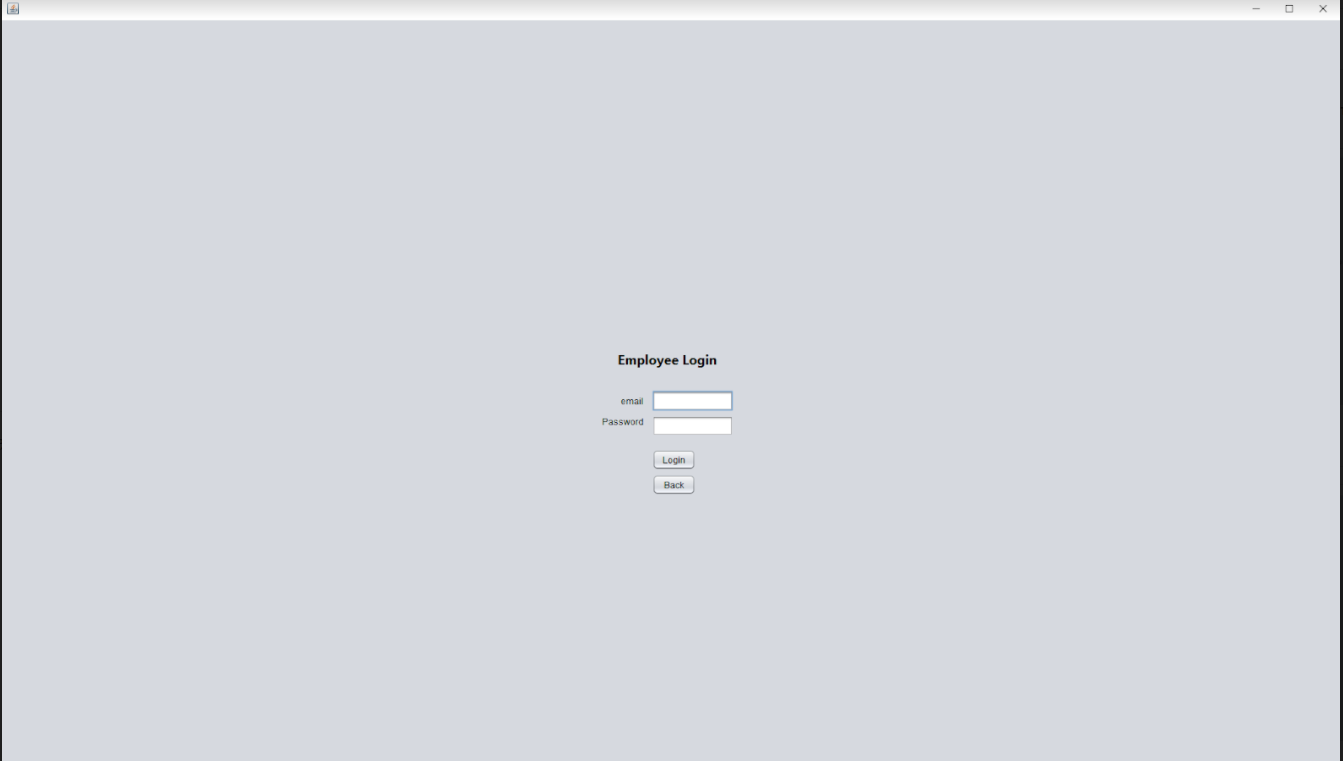
**ΣΗΜΕΙΩΣΗ:** **Τα κουμπιά αυτά έχουν κατασκευαστεί αλλά δεν έχουν υλοποιηθεί και τα παραπάνω αποτελούν τον τρόπο που υποθετικά θα προσπαθούσαμε να τα υλοποιήσουμε αν δεν παρουσιαζόταν θέμα με την σύνδεση της βάσης με την εφαρμογή NetBeans. Ωστόσο θεωρούμε ότι αποτελούν απόδειξη κατανόησης της θεωρίας για την χρήση της JDBC.**

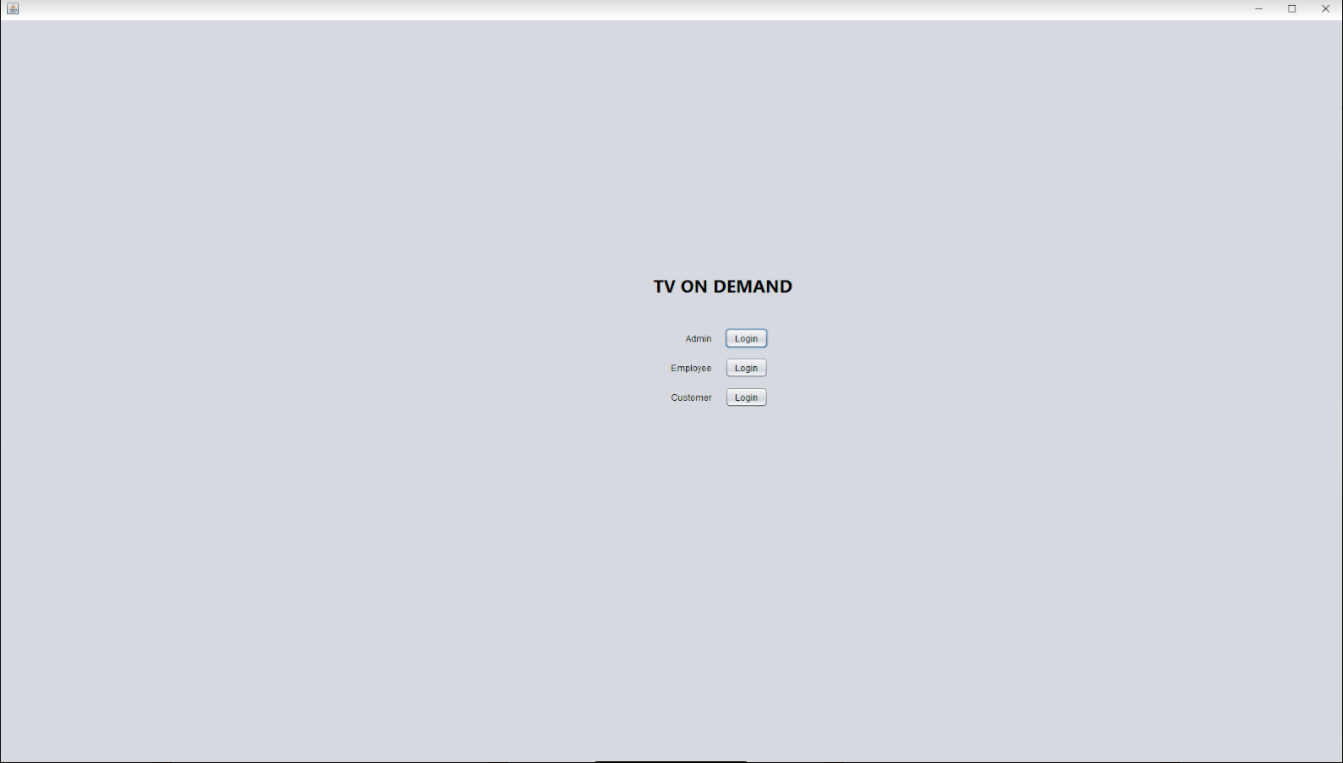
Ενδεικτικά screenshots του GUI:

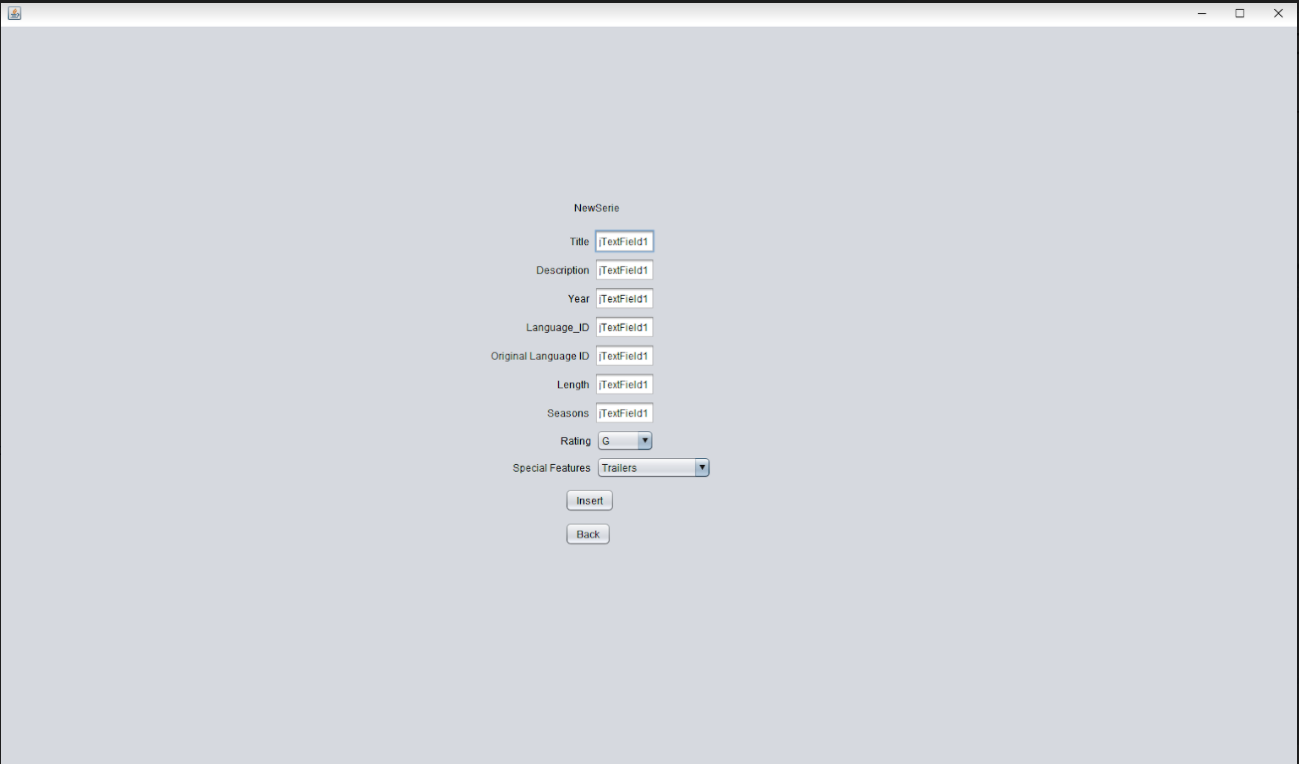
* customerlogin



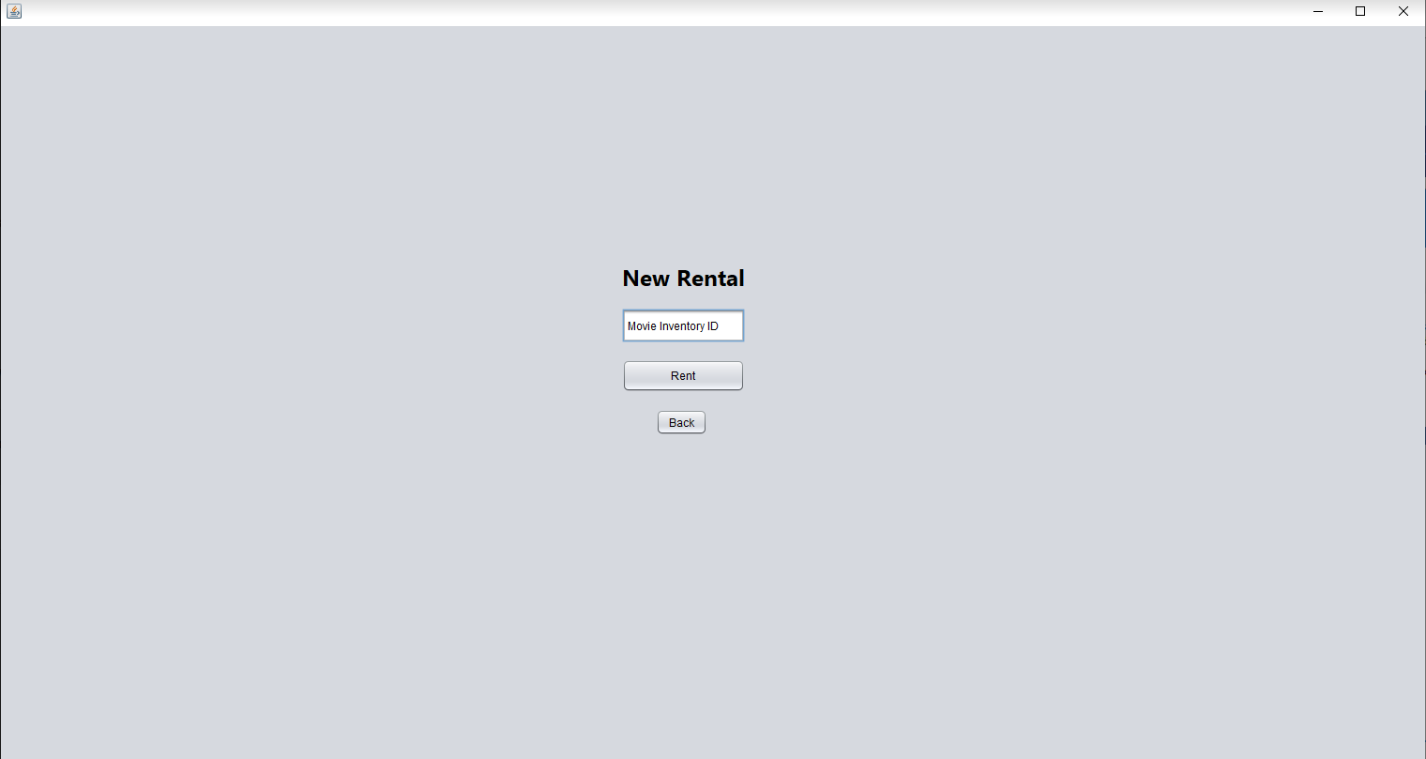
* customerregister
* employeelogin



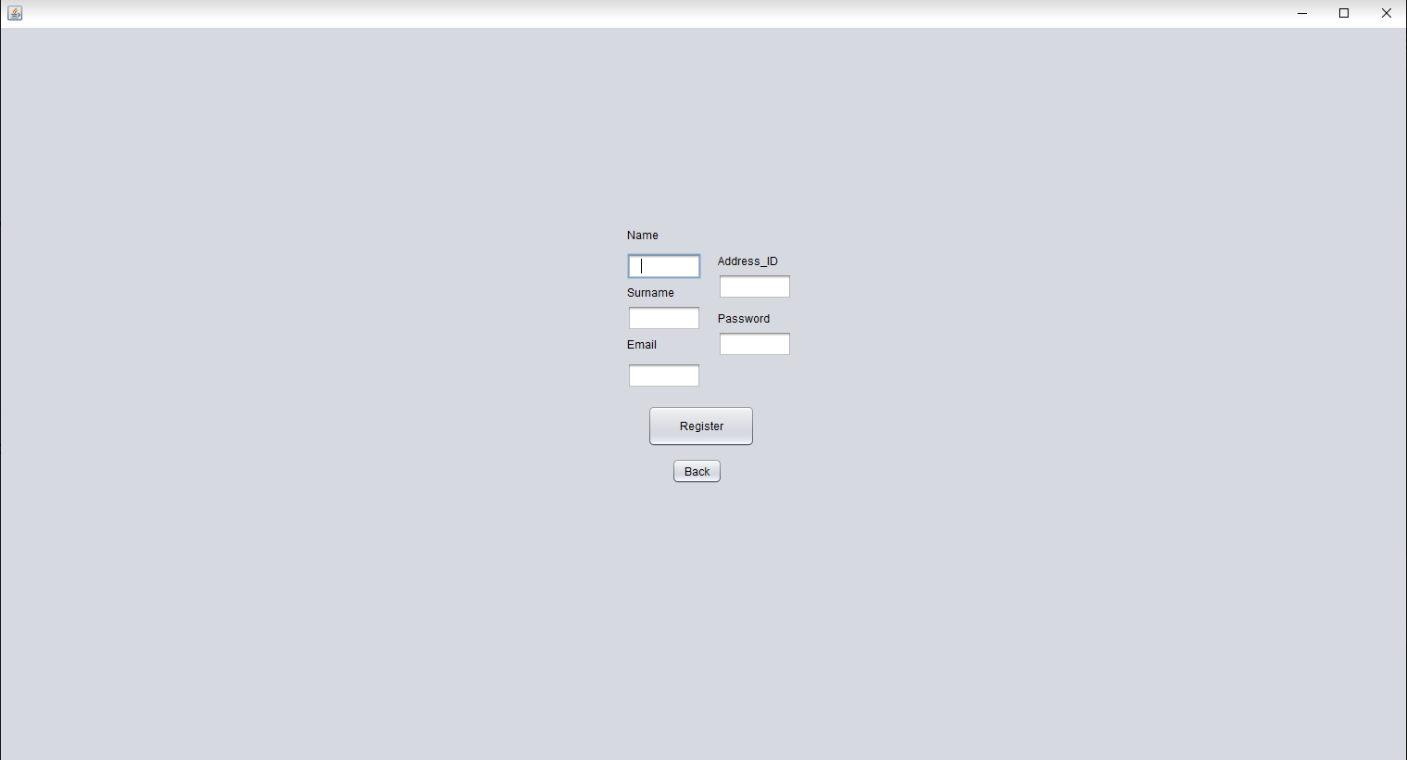
* loginwho
* newserieform



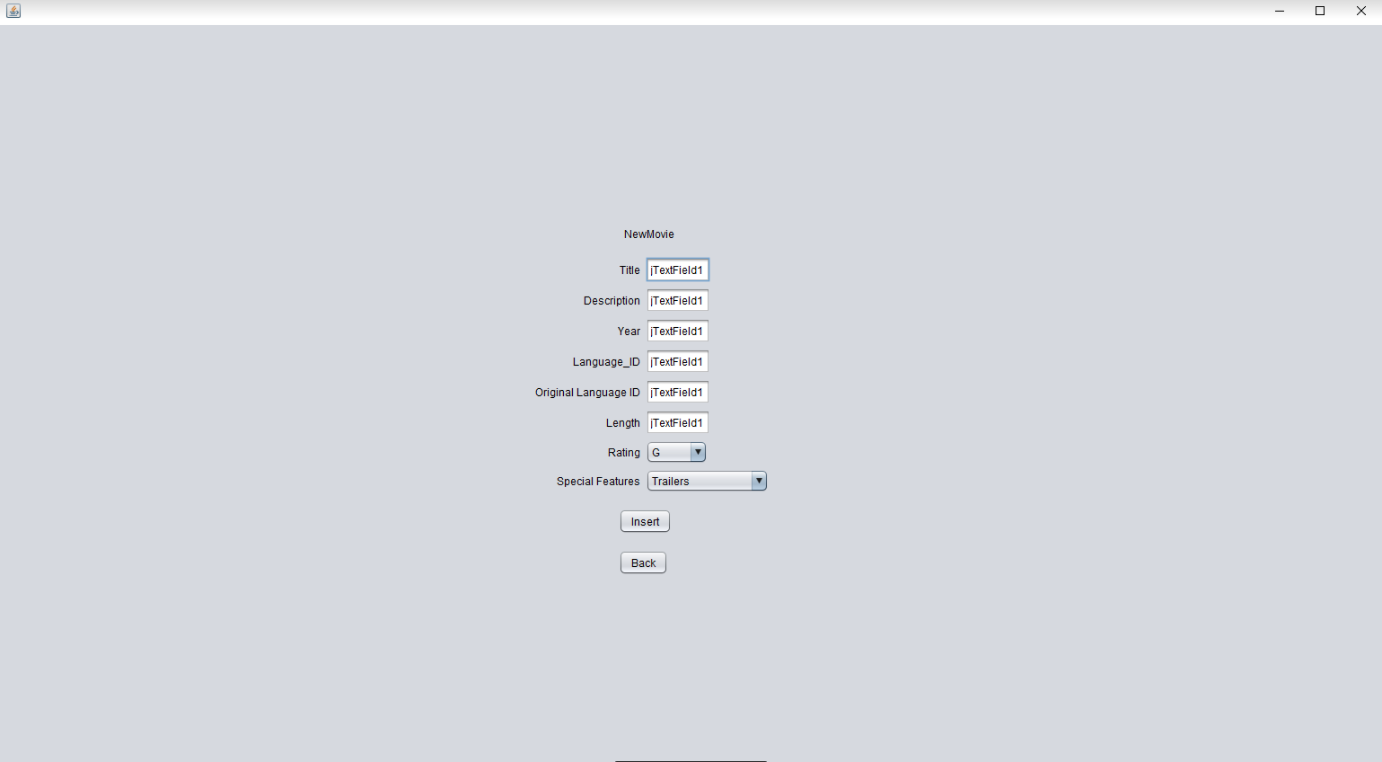
* newrentalform



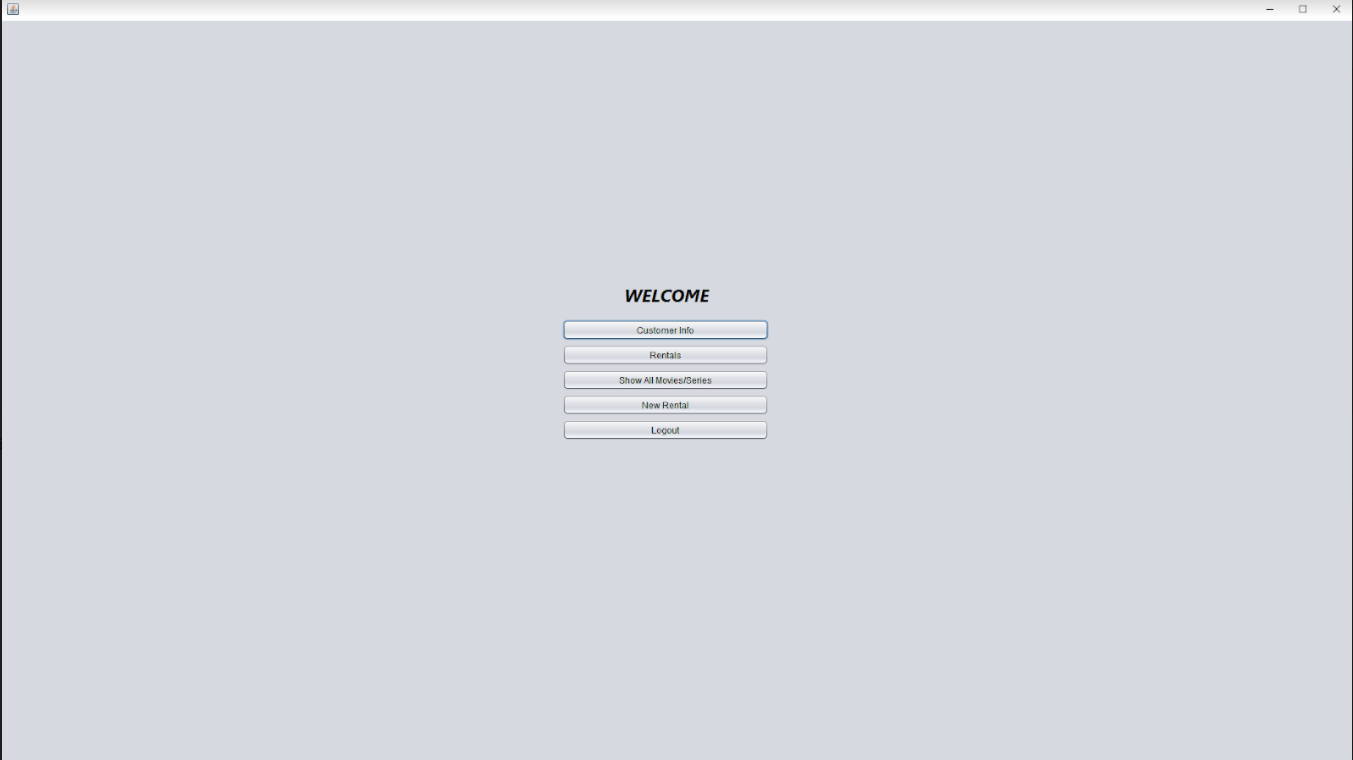
* newemployeeform



* newmovieform



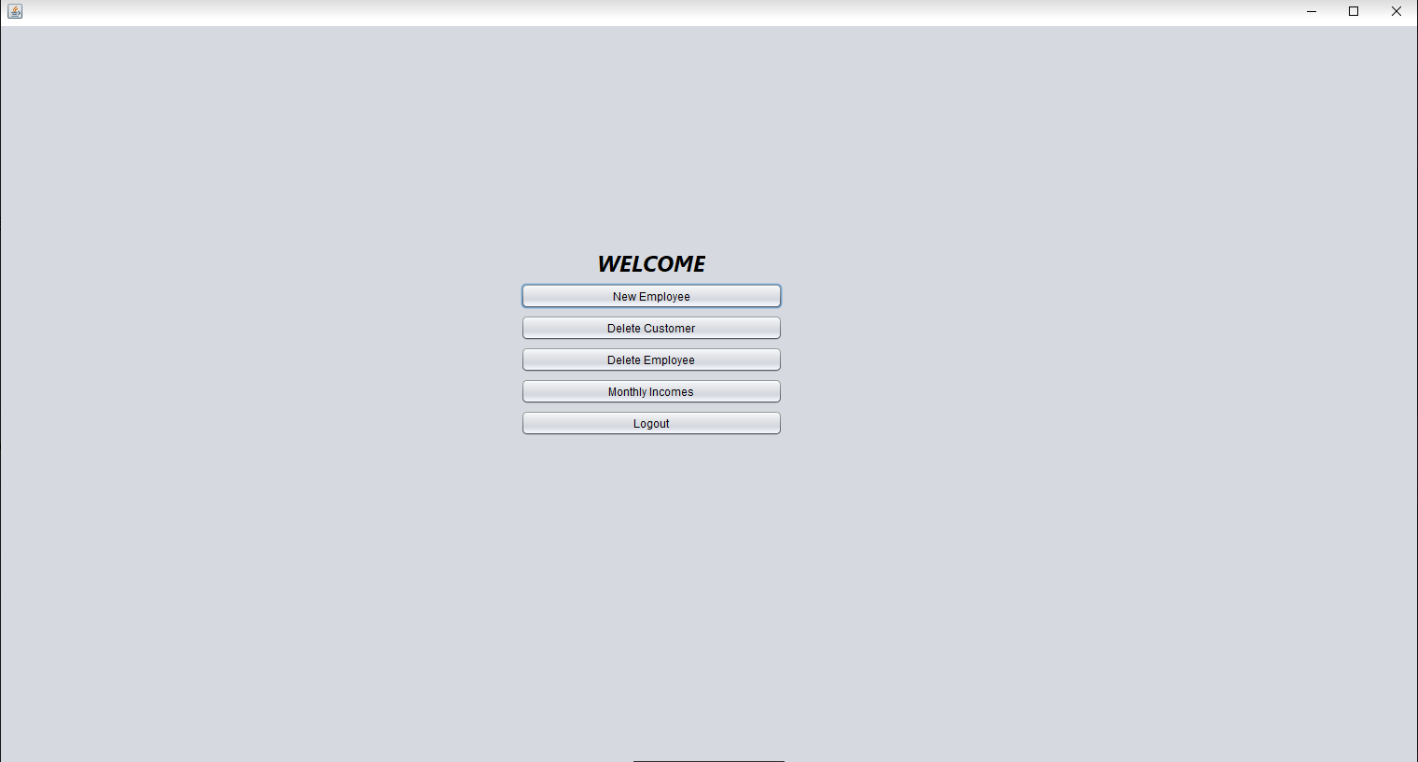
* customermenu



* employeemenu



* adminmenu



* adminlogin

