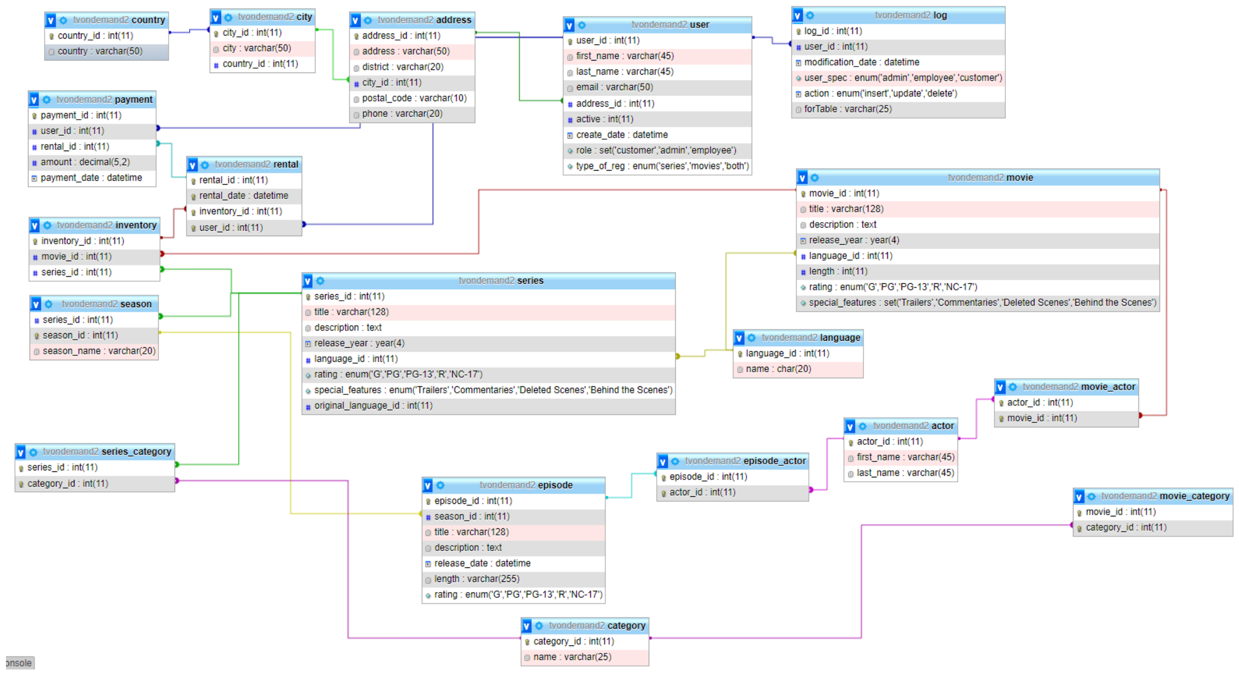
Βάσεις Δεδομένων 2022

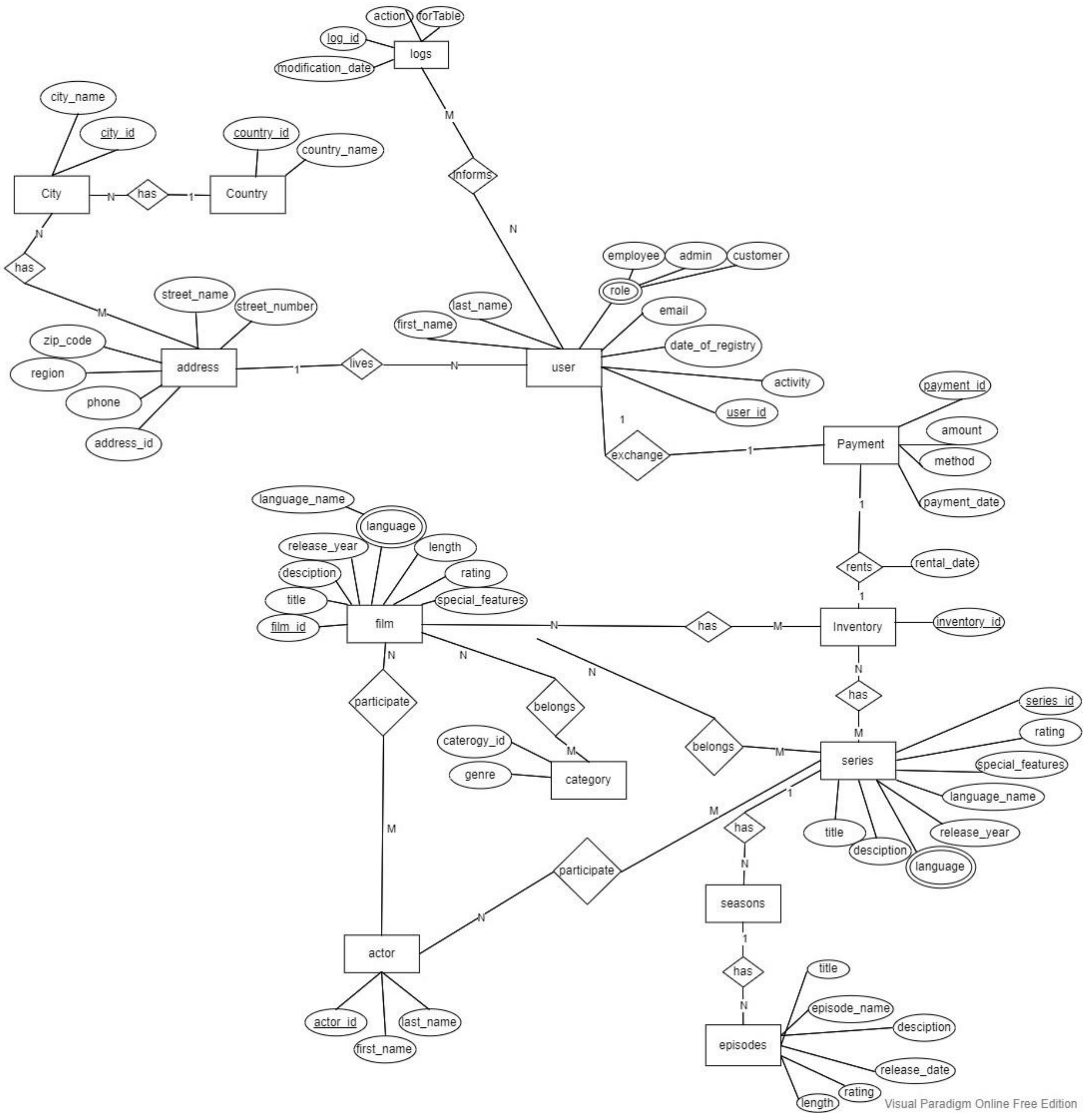
Φοιτητής Νο1 – ΑΜ

Φοιτητής Νο2 – ΑΜ

Σχεσιακό διάγραμμα



ER διάγραμμα



B)

Οι πίνακες που χρειάστηκαν τροποποίηση είναι οι customer και inventory

Ο πίνακας customer άλλαξε σε user και προστέθηκε ένα column με όνομα “role”, όπου δηλώνεται ο ρόλος του κάθε user (customer, admin, employee). Επίσης προστέθηκε άλλο ένα column με όνομα “type_of_reg”, στο οποίο δηλώνεται αν ο χρήστης έκανε εγγραφή για ταινίες, σειρές ή και τα 2

```
DROP TABLE IF EXISTS `user`;
CREATE TABLE IF NOT EXISTS `user` (
  `user_id` int  NOT NULL AUTO_INCREMENT,
  `first_name` varchar(45) NOT NULL,
  `last_name` varchar(45) NOT NULL,
  `email` varchar(50) DEFAULT NULL,
  `address_id` int  NOT NULL,
  `active` int NOT NULL DEFAULT '1',
  `create_date` datetime NOT NULL,
  `role` set('customer','admin','employee') DEFAULT 'customer',
  `type_of_reg` enum('series','movies','both') DEFAULT NULL,
  PRIMARY KEY (`user_id`),
  KEY `fk_customer_address` (`address_id`),
  FOREIGN KEY (`address_id`) REFERENCES `address`(`address_id`)
) ENGINE=InnoDB AUTO_INCREMENT=597 DEFAULT CHARSET=utf8mb4;
```

Στον πίνακα inventory προστέθηκε ένα column με όνομα “series_id”, αφού στο inventory μπορούν να υπάρχουν ταινίες, σειρές ή και τα 2

```
DROP TABLE IF EXISTS `inventory`;
CREATE TABLE IF NOT EXISTS `inventory` (
  `inventory_id` int  NOT NULL AUTO_INCREMENT,
  `movie_id` int  DEFAULT NULL,
  `series_id` int  DEFAULT NULL,
  PRIMARY KEY (`inventory_id`),
  FOREIGN KEY (`movie_id`) REFERENCES `movie`(`movie_id`),
  FOREIGN KEY (`series_id`) REFERENCES `series`(`series_id`)
) ENGINE=InnoDB AUTO_INCREMENT=4421 DEFAULT CHARSET=utf8mb4;
```

Επιπρόσθετοι πίνακες

Προστέθηκαν οι παρακάτω πίνακες, έτσι ώστε να μπορεί να επιτευχθεί το επιθυμητό αποτέλεσμα

Series:

```
DROP TABLE IF EXISTS `series`;
CREATE TABLE IF NOT EXISTS `series` (
 `series_id` int  NOT NULL AUTO_INCREMENT,
 `title` varchar(128) NOT NULL,
 `description` text,
 `release_year` year(4) DEFAULT NULL,
 `language_id` int  NOT NULL,
 `rating` enum('G','PG','PG-13','R','NC-17') DEFAULT 'G',
 `special_features` enum('Trailers','Commentaries','Deleted Scenes','Behind the Scenes') DEFAULT NULL,
 `original_language_id` int  DEFAULT NULL,
 PRIMARY KEY (`series_id`),
 FOREIGN KEY (`language_id`) REFERENCES `language`(`language_id`)
) ENGINE=InnoDB AUTO_INCREMENT=41 DEFAULT CHARSET=utf8mb4;
```

Ένας πίνακας που δηλώνει τις σειρές που θα είναι διαθέσιμες οι οποίες συνδέονται με τον παρακάτω πίνακα “season”

Season

```
DROP TABLE IF EXISTS `season`;
CREATE TABLE IF NOT EXISTS `season` (
 `series_id` int  DEFAULT NULL,
 `season_id` int  NOT NULL,
 `season_name` varchar(20) NOT NULL,
 PRIMARY KEY (`season_id`),
 FOREIGN KEY (`series_id`) REFERENCES `series`(`series_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Αυτός ο πίνακας εμφανίζει τις σεζόν από τις οποίες αποτελούνται οι σειρές

Episode:

```
DROP TABLE IF EXISTS `episode`;
CREATE TABLE IF NOT EXISTS `episode` (
 `episode_id` int  NOT NULL AUTO_INCREMENT,
 `season_id` int  NOT NULL,
 `title` varchar(128) NOT NULL,
 `description` text,
 `release_date` datetime DEFAULT NULL,
 `length` varchar(255) DEFAULT NULL,
 `rating` enum('G','PG','PG-13','R','NC-17') DEFAULT 'G',
 PRIMARY KEY (`episode_id`),
 FOREIGN KEY (`season_id`) REFERENCES `season`(`season_id`)
) ENGINE=InnoDB AUTO_INCREMENT=181 DEFAULT CHARSET=utf8mb4;
```

Σε αυτόν τον πίνακα αποθηκεύονται τα επεισόδια κάθε σεζόν

Episode_actor:

```
DROP TABLE IF EXISTS `episode_actor`;
CREATE TABLE IF NOT EXISTS `episode_actor` (
 `episode_id` int  NOT NULL,
 `actor_id` int  NOT NULL,
 PRIMARY KEY (`episode_id`,`actor_id`),
 FOREIGN KEY (`actor_id`) REFERENCES `actor`(`actor_id`),
 FOREIGN KEY (`episode_id`) REFERENCES `episode`(`episode_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Δηλώνει τους ηθοποιούς που παίζουν στο κάθε επεισόδιο σε μία σειρά (παίρνει δεδομένα από τον πίνακα actor)

Series_category:

```
DROP TABLE IF EXISTS `series_category`;
CREATE TABLE IF NOT EXISTS `series_category` (
 `series_id` int  NOT NULL,
 `category_id` int  NOT NULL AUTO_INCREMENT,
 PRIMARY KEY (`series_id`,`category_id`),
 FOREIGN KEY (`category_id`) REFERENCES `category`(`category_id`),
 FOREIGN KEY (`series_id`) REFERENCES `series`(`series_id`)
) ENGINE=InnoDB AUTO_INCREMENT=12 DEFAULT CHARSET=utf8mb4;
```

Δηλώνει την κατηγορία που ανήκει κάθε σειρά, από τις κατηγορίες στον πίνακα category

Log:

```sql
DROP TABLE IF EXISTS `log`;
CREATE TABLE IF NOT EXISTS `log` (
 `log_id` int  NOT NULL AUTO_INCREMENT,
 `user_id` int,
 `modification_date` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
 `user_spec` enum('admin','employee','customer') CHARACTER SET utf8 DEFAULT 'customer',
 `action` enum('insert','update','delete') CHARACTER SET utf8 NOT NULL,
 `forTable` varchar(25) CHARACTER SET utf8 DEFAULT NULL,
 PRIMARY KEY (`log_id`),
 FOREIGN KEY (`user_id`) REFERENCES `user`(`user_id`)
) ENGINE=InnoDB AUTO_INCREMENT=141 DEFAULT CHARSET=latin1;
```

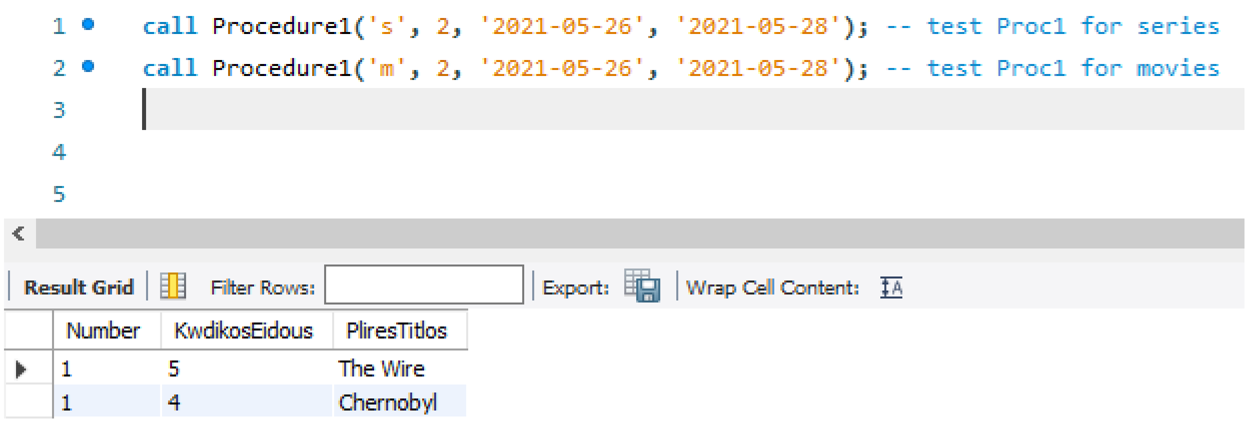
Σε αυτόν τον πίνακα θα καταγράφονται όποιες ενέργειες θέλουμε να αποθηκεύουμε

Stored Procedures

To 1^{o} Procedure

```
DELIMITER $$
--
-- Procedures
--
DROP PROCEDURE IF EXISTS `Procedure1`$$
CREATE PROCEDURE `Procedure1`
(
 IN `charactiras` CHAR(1),
 IN `arithmos` INT,
 IN `begin_date` DATE,
 IN `end_date` DATE
)
BEGIN
 IF (charactiras LIKE 'm')
  THEN
   SELECT COUNT(*) AS Number,
   movie.movie_id AS KwdikosEidous,
   movie.title AS PliresTitlos
    FROM rental
    INNER JOIN inventory ON rental.inventory_id = inventory.inventory_id
     INNER JOIN movie ON inventory.movie_id = movie.movie_id
      WHERE rental.rental_date BETWEEN begin_date AND end_date
     GROUP BY movie.title
     ORDER BY Number DESC
     LIMIT 0, arithmos;
  ELSE SELECT COUNT(*) AS Number,
   series.series_id AS KwdikosEidous,
   series.title AS PliresTitlos
    FROM rental
    INNER JOIN inventory ON rental.inventory_id = inventory.inventory_id
      INNER JOIN series ON inventory.series_id = series.series_id
     WHERE rental.rental_date BETWEEN begin_date AND end_date
     GROUP BY series.title
      ORDER BY Number DESC
      LIMIT 0, arithmos;
   END IF;
END$$
DELIMITER ;
```

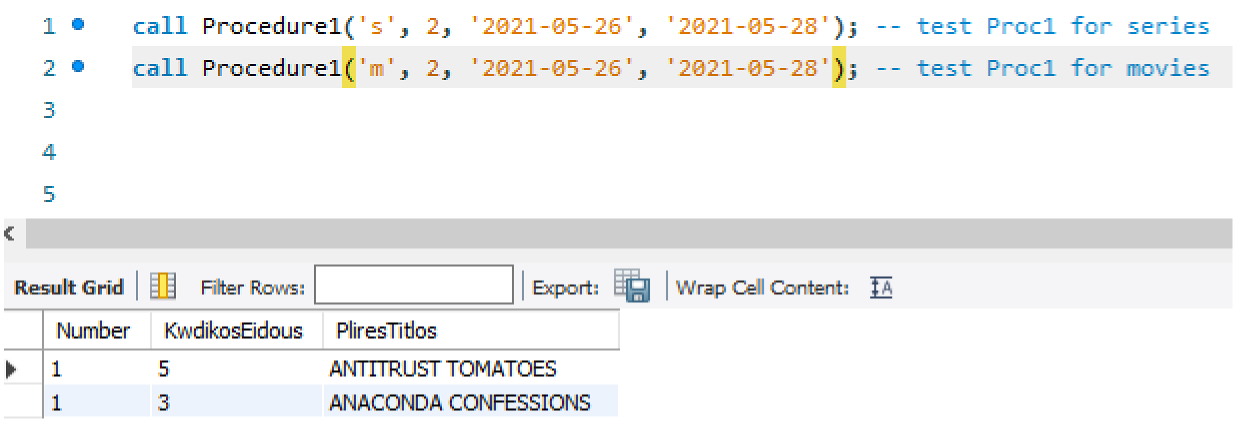
Αρχικά καλούμε το Procedure1, με στόχο σειρές:



```
call Procedure1('s', 2, '2021-05-26', '2021-05-28'); -- test Proc1 for series
call Procedure1('m', 2, '2021-05-26', '2021-05-28'); -- test Proc1 for movies
```

Number	KwdikosEidous	PliresTitlos
1	5	The Wire
1	4	Chernobyl

Αρχικά καλούμε το Procedure1, με στόχο ταινίες:



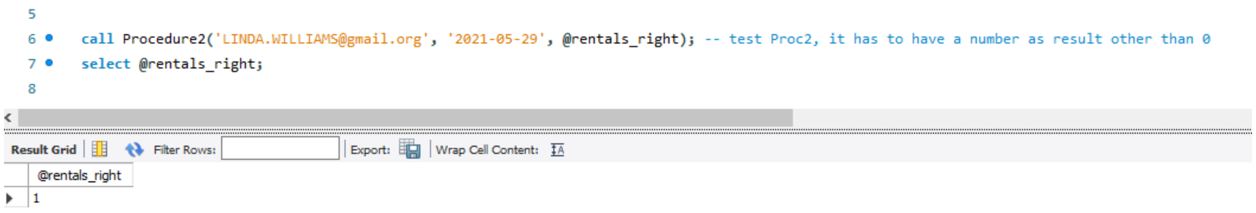
```
call Procedure1('s', 2, '2021-05-26', '2021-05-28'); -- test Proc1 for series
call Procedure1('m', 2, '2021-05-26', '2021-05-28'); -- test Proc1 for movies
```

Number	KwdikosEidous	PliresTitlos
1	5	ANTITRUST TOMATOES
1	3	ANACONDA CONFESSIONS

To 2^{o} Procedure:

```sql
DROP PROCEDURE IF EXISTS `Procedure2`$$
CREATE PROCEDURE `Procedure2`
(
 IN `email` VARCHAR(50),
 IN `Hmerominia` DATE,
   OUT `Rentals` INT
)
BEGIN
 SET `Rentals`= (
  SELECT COUNT(rental_id) AS Rentals
  FROM `rental`
  RIGHT JOIN `user` ON rental.user_id = user.user_id
  WHERE user.email = email
  and DATE(rental.rental_date) LIKE Hmerominia
 );
END$$
```

Παράδειγμα του $2^{ου}$ Procedure:



```sql
call Procedure2('LINDA.WILLIAMS@gmail.org', '2021-05-29', @rentals_right); -- test Proc2, it has to have a number as result other than 0
select @rentals_right;
```

```
@rentals_right
1
```

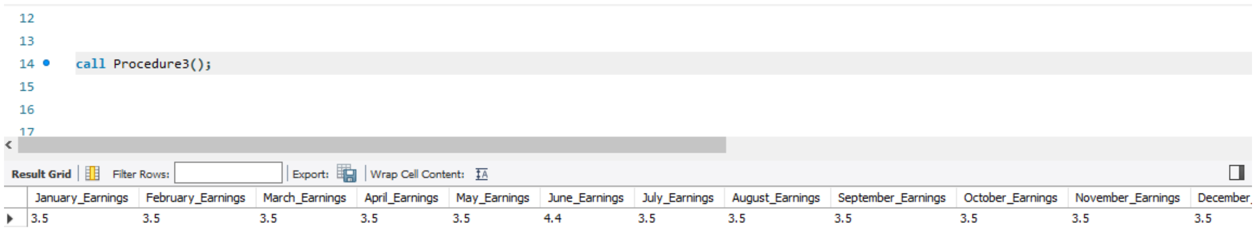
To 3^{o} Procedure:

```
DROP PROCEDURE IF EXISTS Procedure3;
delimiter $$
create procedure Procedure3()
begin
  -- counter = metraei tous mhnes apo ton 1o ews ton 12o
    declare counter int;
    declare movie_count_both int;
    declare movie_count_movies int;
    declare episodes_count_both int;
  declare episodes_count_series int;
    declare earnings float;

    declare January_Earnings float;
    declare February_Earnings float;
    declare March_Earnings float;
    declare April_Earnings float;
    declare May_Earnings float;
    declare June_Earnings float;
    declare July_Earnings float;
    declare August_Earnings float;
    declare September_Earnings float;
    declare October_Earnings float;
    declare November_Earnings float;
    declare December_Earnings float;
    set counter = 0;
    while counter <= 12 do
    -- movie_count_both = to noumero twn atomwn pou exoun enikiasei tainies, kai plhrwnoun 0.3 € ana
tainia
    set movie_count_both = (
      SELECT count(movie.movie_id) as number_of_movies FROM movie
      INNER JOIN inventory ON movie.movie_id=inventory.movie_id
      INNER JOIN rental ON inventory.inventory_id=rental.inventory_id
      WHERE month(rental.rental_date) LIKE counter
            and rental.user_id IN (SELECT user_id from user where type_of_reg = 'both')
    );

    -- movie_count_movies = to noumero twn atomwn pou exoun enikiasei tainies, kai plhrwnoun 0.4 € ana
tainia
        set movie_count_movies = (
      SELECT count(movie.movie_id) as number_of_movies FROM movie
      INNER JOIN inventory ON movie.movie_id=inventory.movie_id
      INNER JOIN rental ON inventory.inventory_id=rental.inventory_id
      WHERE month(rental.rental_date) LIKE counter
            and rental.user_id IN (SELECT user_id from user where type_of_reg = 'movies')
    );

    -- series_count_both = to noumero twn atomwn pou exoun enikiasei seires, kai plhrwnoun 0.1 € ana
epeisodio seiras
```

```
   set episodes_count_both = (
   SELECT count(`episode_id`) from `episode`
   inner join `season` on `episode`.`season_id` = `season`.`season_id`
   inner join `series` on `season`.`series_id` = `series`.`series_id`
   WHERE series.series_id IN (
    SELECT series.series_id AS number_of_series FROM series
    INNER JOIN inventory ON series.series_id = inventory.series_id
    INNER JOIN rental ON inventory.inventory_id = rental.inventory_id
    WHERE month(rental.rental_date) LIKE 5
    and rental.user_id IN (SELECT user_id from user where type_of_reg = 'both')
   )
  );

  -- series_count_series = to noumero twn atomwn pou exoun enikiasei seires, kai plhrwnoun 0.2 € ana
epeisodio seiras
  set episodes_count_series = (
  SELECT count(`episode_id`) from `episode`
  inner join `season` on `episode`.`season_id` = `season`.`season_id`
  inner join `series` on `season`.`series_id` = `series`.`series_id`
  WHERE series.series_id IN (
    SELECT series.series_id AS number_of_series FROM series
    INNER JOIN inventory ON series.series_id = inventory.series_id
    INNER JOIN rental ON inventory.inventory_id = rental.inventory_id
    WHERE month(rental.rental_date) LIKE 5
    and rental.user_id IN (SELECT user_id from user where type_of_reg = 'series')
   )
  );

  set earnings = movie_count_both * 0.3 + movie_count_movies * 0.4 + episodes_count_both * 0.1 +
episodes_count_series * 0.2;


    -- emfanizw ta apotelesmata se mhnes
    IF counter = 0 THEN
   set January_Earnings = (select earnings);
  ELSEIF counter = 1 THEN
   set February_Earnings = (select earnings);
  ELSEIF counter = 2 THEN
   set March_Earnings = (select earnings);
  ELSEIF counter = 3 THEN
   set April_Earnings = (select earnings);
  ELSEIF counter = 4 THEN
   set May_Earnings = (select earnings);
  ELSEIF counter = 5 THEN
   set June_Earnings = (select earnings);
  ELSEIF counter = 6 THEN
   set July_Earnings = (select earnings);
  ELSEIF counter = 7 THEN
```

```
   set August_Earnings = (select earnings);
  ELSEIF counter = 8 THEN
   set September_Earnings = (select earnings);
  ELSEIF counter = 9 THEN
   set October_Earnings = (select earnings);
  ELSEIF counter = 10 THEN
   set November_Earnings = (select earnings);
  ELSEIF counter = 11 THEN
   set December_Earnings = (select earnings);
  END IF;
  set counter = counter + 1;
 end while;
 select January_Earnings, February_Earnings, March_Earnings, April_Earnings, May_Earnings,
June_Earnings, July_Earnings, August_Earnings, September_Earnings, October_Earnings,
November_Earnings, December_Earnings;
end$$
```

Παράδειγμα 3ου Procedure



```
call Procedure3();
```

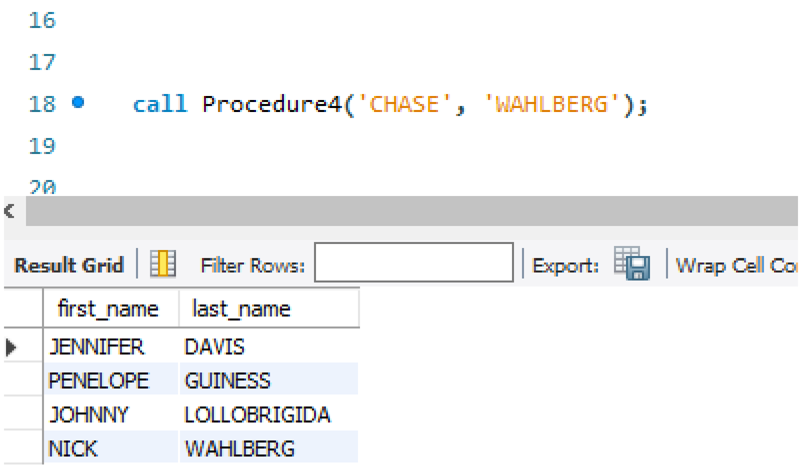
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

January_Earnings	February_Earnings	March_Earnings	April_Earnings	May_Earnings	June_Earnings	July_Earnings	August_Earnings	September_Earnings	October_Earnings	November_Earnings	December
3.5	3.5	3.5	3.5	3.5	4.4	3.5	3.5	3.5	3.5	3.5	3.5

To 4^{o} Procedure:

```
DROP PROCEDURE IF EXISTS `Procedure4`$$
CREATE PROCEDURE `Procedure4`
(
 IN `first_last_name` VARCHAR(45),
 IN `end_last_name` VARCHAR(45)
)
BEGIN
 SELECT count(`actor_id`) as plhthos FROM actor
 WHERE last_name between first_last_name and end_last_name;

 SELECT `first_name`, `last_name` FROM actor
 WHERE last_name between concat(first_last_name, '%') and concat( end_last_name, '%')
 ORDER BY last_name ASC;
END$$
delimiter ;
delimiter $$
DROP PROCEDURE IF EXISTS `Procedure5`$$
CREATE PROCEDURE `Procedure5` (IN `last_name` VARCHAR(45))  BEGIN

 declare count_actors int;
   -- check if there are more than 1 results
 set count_actors = (select count(*) from actor where actor.last_name = last_name);
 IF (count_actors > 1) THEN
 select count_actors as plhthos;
 END IF;
 select  `actor_id`, `first_name`, `last_name` from actor
 where actor.last_name = last_name;
END$$
```

Παράδειγμα του 4^{ου} Procedure



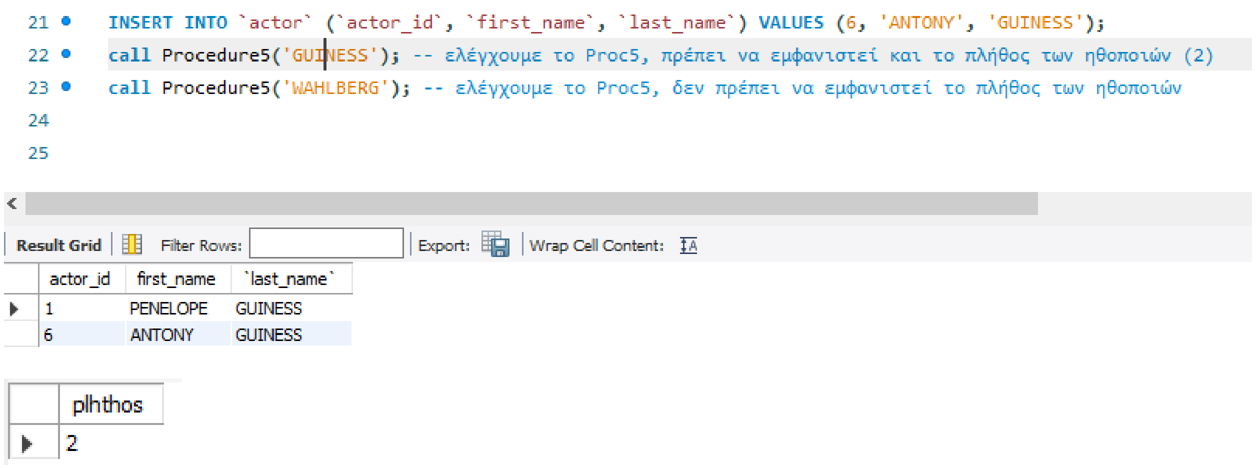
```
call Procedure4('CHASE', 'WAHLBERG');
```

first_name	last_name
JENNIFER	DAVIS
PENELOPE	GUINESS
JOHNNY	LOLLOBRIGIDA
NICK	WAHLBERG

Το 5ο Procedure:

```sql
DROP PROCEDURE IF EXISTS `Procedure5`$$
CREATE PROCEDURE `Procedure5` (IN `last_name` VARCHAR(45))  BEGIN

 declare count_actors int;
   -- check if there are more than 1 results
 set count_actors = (select count(*) from actor where actor.last_name = last_name);
 IF (count_actors > 1) THEN
 select count_actors as plhthos;
 END IF;
 select  `actor_id`, `first_name`, `last_name` from actor
 where actor.last_name = last_name;
END$$
```

Παράδειγμα του 5ου Procedure, στο οποίο πρέπει να βγει και ως αποτέλεσμα και το πλήθος των ηθοποιών:

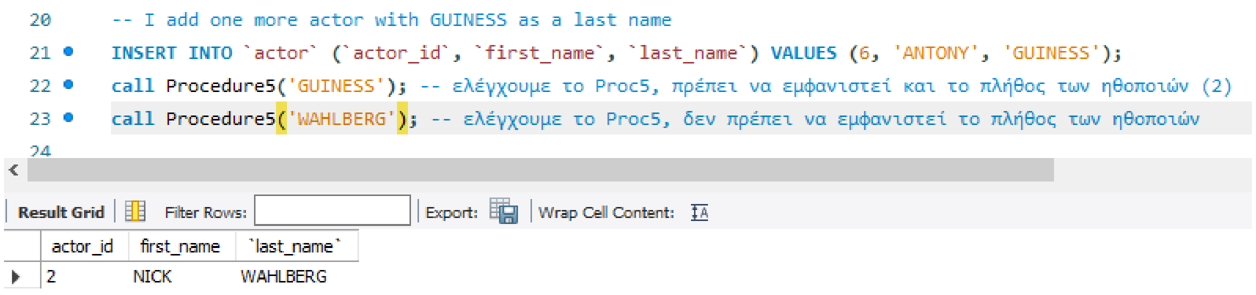


```sql
INSERT INTO `actor` (`actor_id`, `first_name`, `last_name`) VALUES (6, 'ANTONY', 'GUINESS');
call Procedure5('GUINESS'); -- ελέγχουμε το Proc5, πρέπει να εμφανιστεί και το πλήθος των ηθοποιών (2)
call Procedure5('WAHLBERG'); -- ελέγχουμε το Proc5, δεν πρέπει να εμφανιστεί το πλήθος των ηθοποιών
```

actor_id	first_name	`last_name`
1	PENELOPE	GUINESS
6	ANTONY	GUINESS

plhthos
2

Παράδειγμα του 5ου Procedure, στο οποίο δεν πρέπει να βγει και ως αποτέλεσμα και το πλήθος των ηθοποιών:



```sql
-- I add one more actor with GUINESS as a last name
INSERT INTO `actor` (`actor_id`, `first_name`, `last_name`) VALUES (6, 'ANTONY', 'GUINESS');
call Procedure5('GUINESS'); -- ελέγχουμε το Proc5, πρέπει να εμφανιστεί και το πλήθος των ηθοποιών (2)
call Procedure5('WAHLBERG'); -- ελέγχουμε το Proc5, δεν πρέπει να εμφανιστεί το πλήθος των ηθοποιών
```

actor_id	first_name	`last_name`
2	NICK	WAHLBERG

Triggers

Triggers για τον πίνακα inventory:

```
DROP TRIGGER IF EXISTS `update_log_on_inventory_insert`;
DELIMITER $$
CREATE TRIGGER `update_log_on_inventory_insert`
BEFORE INSERT ON `inventory` FOR EACH ROW
BEGIN
  INSERT INTO `log` (user_id, user_spec, action, forTable) VALUES(NULL, 'customer', 'insert', 'inventory');
END
$$
DELIMITER ;

DROP TRIGGER IF EXISTS `update_log_on_inventory_update`;
DELIMITER $$
CREATE TRIGGER `update_log_on_inventory_update` BEFORE UPDATE ON `inventory` FOR EACH ROW
BEGIN
INSERT INTO log(user_id, user_spec, action, forTable) VALUES(NULL, 'customer', 'update', 'inventory') ;
END
$$
DELIMITER ;

DROP TRIGGER IF EXISTS `update_log_on_inventory_delete`;
DELIMITER $$
CREATE TRIGGER `update_log_on_inventory_delete` BEFORE DELETE ON `inventory` FOR EACH ROW
BEGIN
INSERT INTO log(user_id, user_spec, action, forTable) VALUES(NULL, 'customer', 'delete', 'inventory');
END
$$
DELIMITER ;
```

Triggers για τον πίνακα payment:

```
DROP TRIGGER IF EXISTS `update_log_on_payment_insert`;
DELIMITER $$
CREATE TRIGGER `update_log_on_payment_insert` BEFORE INSERT ON `payment` FOR EACH ROW BEGIN
INSERT INTO log(user_id, user_spec, action, forTable) VALUES(NEW.`user_id`, 'customer', 'insert',
'payment');
END
$$
DELIMITER ;
DROP TRIGGER IF EXISTS `update_log_on_payment_update`;
DELIMITER $$
CREATE TRIGGER `update_log_on_payment_update` BEFORE UPDATE ON `payment` FOR EACH ROW
BEGIN
INSERT INTO log(user_id, user_spec, action, forTable) VALUES(NEW.`user_id`, 'customer','update',
'payment') ;
END
$$
DELIMITER ;
DROP TRIGGER IF EXISTS `update_log_on_payment_delete`;
DELIMITER $$
CREATE TRIGGER `update_log_on_payment_delete` BEFORE DELETE ON `payment` FOR EACH ROW
BEGIN
INSERT INTO log(user_id, user_spec, action, forTable) VALUES(OLD.`user_id`, 'customer','delete',
'payment');
END
$$
DELIMITER ;
```

Μέχρι τώρα όλα τα trigger έχουν σχέση με τον πίνακα log κυρίως.

Όταν δηλαδή προσθέτουμε κάποιο entry σε κάποιον από τους πίνακες payment και inventory, αυτό αυτόματα αποθηκεύεται και στον πίνακα log

Triggers για τον πίνακα rental:

```
CREATE TRIGGER `update_log_on_rental_insert` BEFORE INSERT ON `rental` FOR EACH ROW BEGIN

 declare `customer_email` varchar(50);
 declare `customer_reg_type` enum('series','movies','both');
 declare `total_today_rentals` int default 0;

 set `customer_email` = (select email from user where user_id = NEW.`user_id`);
 set `customer_reg_type` = (select type_of_reg from user where user_id = NEW.`user_id`);
  call `Procedure2`(customer_email, NEW.`rental_date`, `total_today_rentals`);

  IF  `total_today_rentals` >= 3 THEN
  -- kalw to custom procedure pou eftiaxa gia thn eisagwgh sto payment
      call Proc_payment_after_rent_insert(`customer_reg_type`, 2, NEW.`user_id`, NEW.`rental_id`,
NEW.`rental_date`);

      -- finally insert into log
  INSERT INTO log(user_id, user_spec, action, forTable) VALUES(NEW.`user_id`, 'customer', 'insert',
'rental');
 ELSE
  -- kalw to custom procedure pou eftiaxa gia thn eisagwgh sto payment
  call Proc_payment_after_rent_insert(`customer_reg_type`, 1, NEW.`user_id`, NEW.`rental_id`,
NEW.`rental_date`);

      -- finally insert into log
  INSERT INTO log(user_id, user_spec, action, forTable) VALUES(NEW.`user_id`, 'customer', 'insert',
'rental');
 END IF;

 INSERT INTO log(user_id, user_spec, action, forTable) VALUES(NEW.`user_id`, 'customer', 'insert',
'rental');
END
$$

CREATE TRIGGER `update_log_on_rental_update` BEFORE UPDATE ON `rental` FOR EACH ROW BEGIN
 INSERT INTO log(user_id, user_spec, action, forTable) VALUES(NEW.`user_id`, 'customer','update',
'rental') ;
END
$$

CREATE TRIGGER `update_log_on_rental_delete` BEFORE DELETE ON `rental` FOR EACH ROW BEGIN
INSERT INTO log(user_id, user_spec, action, forTable) VALUES(OLD.`user_id`, 'customer','delete', 'rental');
END
$$
```

Στα update, delete triggers του πίνακα rental, κάνουμε ότι κάναμε και πριν. Δηλαδή προσθέτουμε στον πίνακα log την αλλαγή ή την διαγραφή.

Στο insert trigger του πίνακα rental όμως, ελέγχουμε αν ο χρήστης που κάνει rent έχει ήδη κάνει σήμερα άλλες 3 φορές και αν έχει κάνει, τότε του μειώνουμε το επόμενο rent στην μισή τιμή (η τιμή αποθηκεύεται στον πίνακα payment). Επίσης αποθηκεύουμε αυτή την πράξη και στον πίνακα log

Triggers για τον πίνακα rental:

```
DROP TRIGGER IF EXISTS `DENIED_user_updates`;
DELIMITER $$
CREATE TRIGGER `DENIED_user_updates` BEFORE UPDATE ON `user` FOR EACH ROW BEGIN
 IF (old.`user_id` <> new.`user_id`) THEN
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'User Id cannot change!!';
 ELSEIF (old.`first_name` <> new.`first_name`) THEN
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'A user first name cannot change!!';
  ELSEIF (old.`last_name` <> new.`last_name`) THEN
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'A user last name cannot change!!';
 ELSEIF (old.`create_date` <> new.`create_date`) THEN
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'A user created date cannot change!!';
  END IF;
END
$$
DELIMITER ;

DROP TRIGGER IF EXISTS `update_log_on_user_insert`;
DELIMITER $$
CREATE TRIGGER `update_log_on_user_insert` BEFORE INSERT ON `user` FOR EACH ROW BEGIN
INSERT INTO log(user_id, user_spec, action, forTable) VALUES(NEW.user_id, 'customer', 'insert', 'user');
END
$$
DELIMITER ;

DROP TRIGGER IF EXISTS `update_log_on_user_update`;
DELIMITER $$
CREATE TRIGGER `update_log_on_user_update` BEFORE UPDATE ON `user` FOR EACH ROW BEGIN
INSERT INTO log(user_id, user_spec, action, forTable) VALUES(OLD.user_id, 'customer','update', 'user');
END
$$
DELIMITER ;

DROP TRIGGER IF EXISTS `update_log_on_user_delete`;
DELIMITER $$
CREATE TRIGGER `update_log_on_user_delete` BEFORE DELETE ON `user` FOR EACH ROW BEGIN
INSERT INTO log(user_id, user_spec, action, forTable) VALUES(OLD.user_id, 'customer','delete', 'user');
END
$$
DELIMITER ;
```

Τέλος, στον πίνακα user, πάλι αποθηκεύουμε οποιαδήποτε αλλαγή οποιαδήποτε user στον πίνακα log, αλλά επίσης, δεν αφήνουμε τον χρήστη να κάνει αλλαγές που δεν έπρεπε να κάνει. Δηλαδή, δεν τον αφήνουμε να αλλάξει το “user_id” του, το “first_name” του, το “last_name” του και την ημερομηνία που δημιουργήθηκε.