

# ΑΝΑΦΟΡΑ TVONDEMAND

## ΕΙΣΑΓΩΓΗ

Στην παρούσα αναφορά παρουσιάζουμε:

- Όλους τους νέους πίνακες καθώς και τις τροποποιήσεις που κάναμε στους πίνακες που είχαν δοθεί, για την καλύτερη λειτουργία του προγράμματος.
- Τους κώδικες και τα παραδείγματα για τις Stored Procedures.
- Τους κώδικες και τα παραδείγματα για τα Triggers.

## ΤΕΧΝΟΛΟΓΙΕΣ ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΑΝ ΓΙΑ ΤΟ PROJECT

- WampServer
- MySQL Workbench
- Visual Studio Code
- PHPMYAdmin
- Sublime Text
- Apache NetBeans
- Eclipse

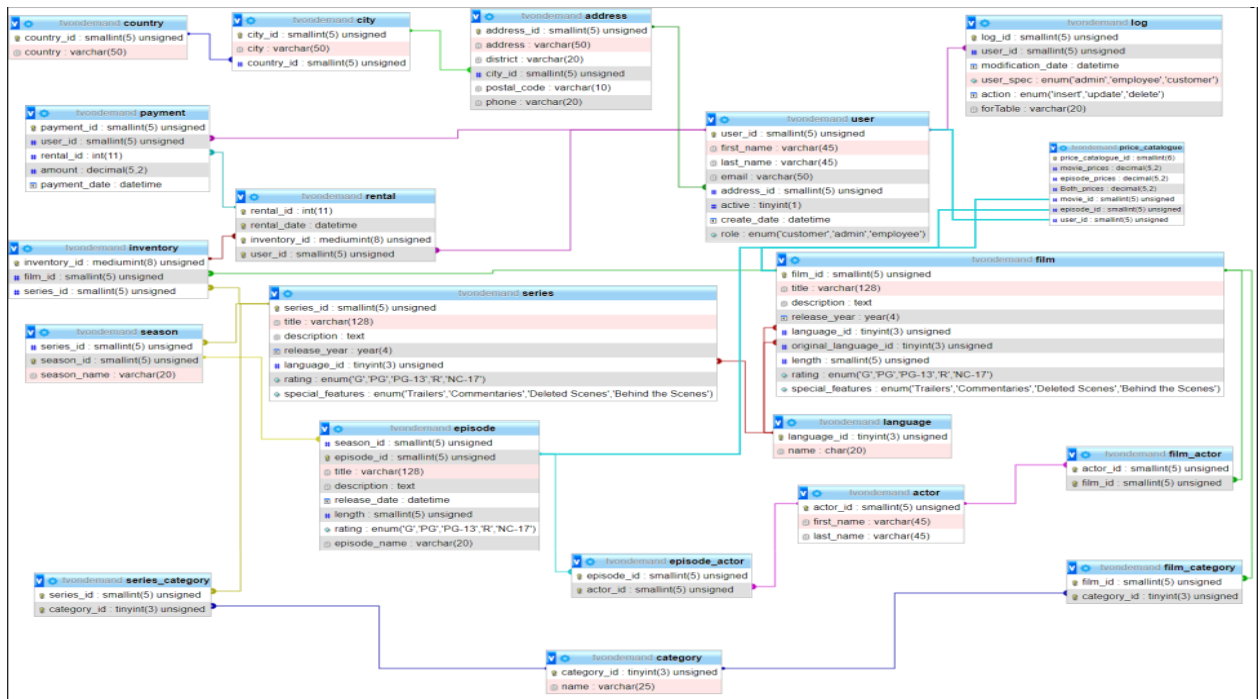
## ΚΕΦΑΛΑΙΟ 1

Σε αυτό το κεφάλαιο δίνονται το τελικό σχεσιακό διάγραμμα και η περιγραφή με το πως χειριστήκαμε τους ήδη υπάρχοντες πίνακες.

### A)

Στο παρακάτω σχήμα φαίνεται το σχεσιακό διάγραμμα της συνολικής Βάσης Δεδομένων.

Δίνεται και ως αρχείο PNG στον φάκελο FINISHED μαζί με το ER (επίσης ως αρχείο PNG).



B)

Στους πίνακες: actor, country, city, address, category, language, movie, movie\_actor, movie\_category, rental και payment δεν έγινε καμία τροποποίηση και βρίσκονται όλοι μαζί στον φάκελο Create-Insert στο αρχείο Starting\_DB.

Οι πίνακες που χρειάστηκαν τροποποίηση είναι οι: customer και inventory.

Τον πίνακα customer τον αλλάξαμε σε πίνακα user και του προσθέσαμε το column 'role' με SET τους ρόλους που μπορεί να έχει ένας user όπως customer, admin και employee με το σκεπτικό ότι ένας administrator μπορεί να είναι και customer και ένας employee το ίδιο. Επίσης του προσθέσαμε ένα column 'type\_of\_reg' για τον τύπο εγγραφής, αν δηλαδή έκανε εγγραφή για να βλέπει μόνο σειρές, μόνο ταινίες ή και τα δυο.

```
DROP TABLE IF EXISTS `user`;  
CREATE TABLE IF NOT EXISTS `user` (  
  `user_id` smallint(5) UNSIGNED NOT NULL AUTO_INCREMENT,  
  `first_name` varchar(45) NOT NULL,  
  `last_name` varchar(45) NOT NULL,  
  `email` varchar(50) DEFAULT NULL,  
  `address_id` smallint(5) UNSIGNED NOT NULL,  
  `active` tinyint(1) 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`)  
);
```

Στον πίνακα inventory απλά προσθέσαμε το column series\_id καθώς σε ένα inventory μπορεί να υπάρχουν ή ταινίες ή σειρές ή και τα δυο.

```
DROP TABLE IF EXISTS `inventory`;  
CREATE TABLE IF NOT EXISTS `inventory` (  
  `inventory_id` mediumint(8) UNSIGNED NOT NULL AUTO_INCREMENT,  
  `movie_id` smallint(5) UNSIGNED DEFAULT NULL,  
  `series_id` smallint(5) UNSIGNED DEFAULT NULL,  
  PRIMARY KEY (`inventory_id`),  
  KEY `fk_inventory_movie` (`movie_id`),  
  KEY `fk_inventory_series` (`series_id`)  
);
```

Αυτοί οι δυο τροποποιημένοι πίνακες δίνονται μαζί με τους νέους πίνακες.

## ΚΕΦΑΛΑΙΟ 2

Σε αυτό το κεφάλαιο δίνεται η περιγραφή των νέων πινάκων που χρειάστηκε να φτιάξουμε για την υλοποίηση του Project.

-- Δημιουργία πίνακα 'series'

```
DROP TABLE IF EXISTS `series`;
CREATE TABLE IF NOT EXISTS `series` (
  `series_id` smallint(5) UNSIGNED NOT NULL AUTO_INCREMENT,
  `title` varchar(128) NOT NULL,
  `description` text,
  `release_year` year(4) DEFAULT NULL,
  `language_id` tinyint(3) UNSIGNED 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` tinyint(3) UNSIGNED DEFAULT NULL,
  PRIMARY KEY (`series_id`),
  KEY `fk_series_language` (`language_id`);
```

- ♦ Παρόμοιος με τον πίνακα movie χωρίς το column 'length' καθώς αυτό το προσθέσαμε στον πίνακα 'episode'.

-- Δημιουργία πίνακα 'season'

```
DROP TABLE IF EXISTS `season`;
CREATE TABLE IF NOT EXISTS `season` (
  `series_id` smallint(5) UNSIGNED DEFAULT NULL,
  `season_id` smallint(5) UNSIGNED NOT NULL,
  `season_name` varchar(20) NOT NULL,
  PRIMARY KEY (`season_id`),
  KEY `fk_season_series` (`series_id`);
```

- ♦ Πίνακας 'season' που δείχνει τους κύκλους μιας σειράς

-- Δημιουργία πίνακα 'episode'

```
DROP TABLE IF EXISTS `episode`;
CREATE TABLE IF NOT EXISTS `episode` (
  `episode_id` int(11) NOT NULL AUTO_INCREMENT,
  `season_id` int(11) 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`),
  KEY `fk_episode_season` (`season_id`);
```

- ♦ Πίνακας 'episode' που δείχνει τα επεισόδια που έχει κάθε κύκλος. Σε αυτό το table προσθέσαμε το length.

--Δημιουργία πίνακα 'episode\_actor'

```
DROP TABLE IF EXISTS `episode_actor`;  
CREATE TABLE IF NOT EXISTS `episode_actor` (  
  `episode_id` smallint(5) UNSIGNED NOT NULL,  
  `actor_id` smallint(5) UNSIGNED NOT NULL,  
  PRIMARY KEY (`episode_id`, `actor_id`),  
  KEY `fk_episode_actor_actor` (`actor_id`)  
);
```

- ♦ Πίνακας 'episode\_actor' που δείχνει τους ηθοποιούς και τις τα επεισόδια στα οποία παίζουν.

-- Δημιουργία πίνακα 'series\_category'

```
DROP TABLE IF EXISTS `series_category`;  
CREATE TABLE IF NOT EXISTS `series_category` (  
  `series_id` smallint(5) UNSIGNED NOT NULL,  
  `category_id` tinyint(3) UNSIGNED NOT NULL AUTO_INCREMENT,  
  PRIMARY KEY (`series_id`, `category_id`),  
  KEY `fk_series_category_category` (`category_id`)  
);
```

- ♦ Πίνακας 'series\_category' που δείχνει σε τι κατηγορία ανήκει κάθε σειρά

--Δημιουργία πίνακα 'log'

```
DROP TABLE IF EXISTS `log`;  
CREATE TABLE IF NOT EXISTS `log` (  
  `log_id` smallint(5) UNSIGNED NOT NULL AUTO_INCREMENT,  
  `user_id` smallint(5) UNSIGNED NOT NULL,  
  `modification_date` datetime NOT NULL,  
  `user_spec` enum('admin', 'employee', 'customer') DEFAULT 'customer',  
  `action` enum('insert', 'update', 'delete') NOT NULL,  
  `forTable` varchar(20) DEFAULT NULL,  
  PRIMARY KEY (`log_id`),  
  KEY `fk_log_user` (`user_id`)  
);
```

- ♦ Πίνακας 'log' στον οποίο καταγράφονται οι ενέργειες που εκτελούνται από οποιονδήποτε χρήστη (user\_id και user\_spec) και σε ποιόν πίνακα εκτελούνται (forTable) καθώς και πότε ακριβώς έγινε η ενέργεια.

-- Δημιουργία πίνακα 'price\_list'

```
DROP TABLE IF EXISTS `price_list`;  
CREATE TABLE IF NOT EXISTS `price_list` (  
  `price_list_id` smallint(6) NOT NULL AUTO_INCREMENT,  
  `movie_prices` decimal(5,2) DEFAULT NULL,  
  `episode_prices` decimal(5,2) DEFAULT NULL,  
  `Both_prices` decimal(5,2) DEFAULT NULL,  
  `movie_id` smallint(5) UNSIGNED NOT NULL,  
  `episode_id` smallint(5) UNSIGNED NOT NULL,  
  `user_id` smallint(5) UNSIGNED NOT NULL,  
  PRIMARY KEY (`price_list_id`),  
  KEY `fk_price_list_movie` (`movie_id`),  
  KEY `fk_price_list_user` (`user_id`),  
  KEY `fk_price_list_series` (`episode_id`)  
);
```

Πίνακας price\_list ο οποίος αποτελεί το μενού και στον οποίο καταγράφεται το ποσό της ταινίας (0,4), το ποσό του επεισοδίου (0,3) αλλά και το ποσό και των δυο μαζί (0,4 αφού είναι 0,3 ανά ταινία και 0,1 ανά επεισόδιο).

## ΚΕΦΑΛΑΙΟ 3

Σε αυτό το κεφάλαιο δίνονται οι Stored Procedures

```
-- 3.1 procedure
-- CALL Procedure1('m', 4, '2002-10-05', '2011-05-09');
-- CALL Procedure1('s', 2, '2002-10-05', '2011-05-09');

DROP PROCEDURE IF EXISTS Procedure1;

DELIMITER $$

CREATE PROCEDURE Procedure1
(
  IN caractiras char(1),
  IN arithmos INT,
  IN begin_date DATE,
  IN end_date DATE
)
BEGIN
  IF (caractiras 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 ;

--

-- 3.2 procedure

--

-- call Procedure2('s','APRIL.BURNS@sakilacustomer.org', '2005-06-20');

DROP PROCEDURE IF EXISTS Procedure2;

DELIMITER $$

CREATE PROCEDURE Procedure2

(

IN email VARCHAR(50),

IN Hmerominia DATE

)

BEGIN

SELECT COUNT (rental_id) AS Rentals

FROM rental

INNER JOIN user ON rental.user_id = user.user_id

WHERE user.email = email

and DATE(rental.rental_date) LIKE Hmerominia;

END$$

DELIMITER ;

--

-- 3.3 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;
set counter = 0;
while counter <= 12 do
-- movie_count_both = to noumero tw n 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 tw n 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 tw n 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 tw n 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  
select earnings as January_Earnings;  
ELSE IF (counter = 1) THEN  
select earnings as February_Earnings;  
ELSE IF (counter = 2) THEN  
select earnings as March_Earnings;  
ELSE IF (counter = 3) THEN  
select earnings as April_Earnings;  
ELSE IF (counter = 4) THEN  
select earnings as May_Earnings;  
ELSE IF (counter = 5) THEN  
select earnings as June_Earnings;  
ELSE IF (counter = 6) THE  
select earnings as July_Earnings;  
ELSE IF (counter = 7) THEN  
select earnings as August_Earnings;  
ELSE IF (counter = 8) THEN  
select earnings as September_Earnings;
```



```

ELSE IF (counter = 9) THEN
select earnings as October_Earnings;
ELSE IF (counter = 10) THEN
select earnings as November_Earnings;
ELSE IF (counter = 11) THEN
select earnings as December_Earnings;
END IF;

set counter = counter +1;

end while;

end$$

-- 3.4.a procedure
--
DROP PROCEDURE IF EXISTS Procedure4;
DELIMITER $$
CREATE PROCEDURE Procedure4
(
IN first_last_name VARCHAR(45),
IN end_last_name VARCHAR(45)
)
BEGIN
SELECT count(`actor_id`) as plhthos FROM actor
WHERE last_name between first_last_name and end_last_name;
SELECT `first_name`, `last_name` FROM actor
WHERE last_name between concat(first_last_name, '%') and concat( end_last_name, '%')
ORDER BY last_name ASC;
END$$
DELIMITER ;

--call Procedure4('Aco', 'Alm');

-- for 3.4.b

-- make an index first
--
ALTER TABLE `actor` ADD INDEX actor_index_last_name(last_name);
--

-- 3.4.b procedure
--
DROP PROCEDURE IF EXISTS Procedure5;

```

```

DELIMITER $$
CREATE PROCEDURE Procedure5
(
IN last_name VARCHAR(45)
)
BEGIN
    declare count_actors int;
    -- check if there are more than 1 results
    set count_actors = (select count(*) from actor where actor.last_name = last_name);
    IF (count_actors > 1) THEN
select count_actors as plhthos;
    END IF;
    select `actor_id`, `first_name`, `last_name` from actor
    where actor.last_name = last_name;
END$$
DELIMITER ;

-- 1 result
-- call Procedure4('GUINNESS');
-- more than 1 results
-- call Procedure4('Livingston');

```

✓ Showing rows 0 - 3 (4 total, Query took 0.0016 seconds.)

`CALL Procedure1('m', 4, '2002-10-05', '2011-05-09');`

[\[ Edit inline \]](#) [\[ Edit \]](#) [\[ Create PHP code \]](#)




☐ Show all | Number of rows: 25 | Filter rows:

+ Options

Number	KwdikosEidous	PliresTitlos
3	379	GREEDY ROOTS
3	859	SUGAR WONKA
3	902	TRADING PINOCCHIO
3	914	TROUBLE DATE

☐ Show all | Number of rows: 25 | Filter rows:

Query results operations

 Print  Copy to clipboard  Create view

✓ Showing rows 0 - 1 (2 total, Query took 0.0008 seconds.)

`CALL Procedure1('s', 2, '2002-10-05', '2011-05-09');`

[\[ Edit inline \]](#) [\[ Edit \]](#) [\[ Create PHP code \]](#)




☒ Show all | Number of rows: All | Filter rows:

+ Options

Number	KwdikosEidous	PliresTitlos
7	35	The Vietnam War
7	29	Game of Thrones

☒ Show all | Number of rows: All | Filter rows:

Query results operations

 Print  Copy to clipboard  Create view



## ΚΕΦΑΛΑΙΟ 4

Σε αυτό το κεφάλαιο δίνονται τα Triggers.

```
-- trigger1
-- user
--
delimiter //
CREATE TRIGGER enimerwsi_log_1_1
BEFORE INSERT ON user
FOR EACH ROW
BEGIN
INSERT INTO log(user_spec, action, forTable) VALUES('customer', 'insert', 'user');
END//
--
CREATE TRIGGER enimerwsi_log_1_2
BEFORE UPDATE ON user
FOR EACH ROW
BEGIN
INSERT INTO log(user_spec, action, forTable) VALUES('customer','update', 'user');
END//
--
CREATE TRIGGER enimerwsi_log_1_3
BEFORE DELETE ON user
FOR EACH ROW
BEGIN
INSERT INTO log(user_spec, action, forTable) VALUES('customer','delete', 'user');
END//
--
-- rental
--
CREATE TRIGGER enimerwsi_log_1_4
BEFORE INSERT ON rental
FOR EACH ROW
BEGIN
```

```

INSERT INTO log(user_spec, action, forTable) VALUES('customer', 'insert', 'rental');
END//

--

CREATE TRIGGER enimerwsi_log_1_5
BEFORE UPDATE ON rental
FOR EACH ROW
BEGIN
INSERT INTO log(user_spec, action, forTable) VALUES('customer','update', 'rental') ;
END//

--

CREATE TRIGGER enimerwsi_log_1_6
BEFORE DELETE ON rental
FOR EACH ROW
BEGIN
INSERT INTO log(user_spec, action, forTable) VALUES('customer','delete', 'rental');
END//

--

-- inventory
--

CREATE TRIGGER enimerwsi_log_1_7
BEFORE INSERT ON inventory
FOR EACH ROW
BEGIN
INSERT INTO log(user_spec, action, forTable) VALUES('customer', 'insert', 'inventory');
END//

--

CREATE TRIGGER enimerwsi_log_1_8
BEFORE UPDATE ON inventory
FOR EACH ROW
BEGIN
INSERT INTO log(user_spec, action, forTable) VALUES('customer','update', 'inventory') ;
END//

--

CREATE TRIGGER enimerwsi_log_1_9
BEFORE DELETE ON inventory
FOR EACH ROW

```

```

BEGIN
INSERT INTO log(user_spec, action, forTable) VALUES('customer','delete', 'inventory');
END//
--
-- payment
--
CREATE TRIGGER enimerwsi_log_1_10
BEFORE INSERT ON payment
FOR EACH ROW
BEGIN
INSERT INTO log(user_spec, action, forTable) VALUES('customer', 'insert', 'payment');
END//
--
CREATE TRIGGER enimerwsi_log_1_11
BEFORE UPDATE ON payment
FOR EACH ROW
BEGIN
INSERT INTO log(user_spec, action, forTable) VALUES('customer','update', 'payment') ;
END//
--
CREATE TRIGGER enimerwsi_log_1_12
BEFORE DELETE ON payment
FOR EACH ROW
BEGIN
INSERT INTO log(user_spec, action, forTable) VALUES('customer','delete', 'payment');
END//
delimiter ;
--
-- StoredProcedure gia to trigger2
--
DROP PROCEDURE IF EXISTS ProcedureEkptwsi;
DELIMITER $$
CREATE PROCEDURE ProcedureEkptwsi
(
IN Username VARCHAR(25),
IN Hmerominia DATE

```

```

)
BEGIN
SELECT COUNT (rental_id) AS Enoikiaseis
FROM rental
INNER JOIN user ON rental.user_id=user.user_id
WHERE rental.rental_date=CURRENT_TIMESTAMP
GROUP BY user.email;
IF Enoikiaseis>=3 THEN
SELECT amount FROM rental INNER JOIN payment
ON rental.rental_id=payment.rental_id;
    UPDATE payment.amount SET payment.amount=payment.default_price/2;
ELSE
SELECT updated_amount FROM rental INNER JOIN payment
    ON rental.rental_id=payment.rental_id;
    UPDATE payment.amount SET payment.amount=payment.default_price;
END IF;
END;
END$$
--
-- trigger2
--
DELIMITER #
CREATE TRIGGER EKPTWSH
BEFORE UPDATE ON rental FOR EACH ROW
BEGIN
    CALL ProcedureEkptwsi;
END;
END#
DELIMITER ;
--
--trigger3
--
DROP PROCEDURE IF EXISTS DENIED_1;
DELIMITER $$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$$

```

- ♦ Τα triggers δίνονται και στον φάκελο Triggers σε μορφή .SQL

## ΙΣΤΟΣΕΛΙΔΕΣ ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΑΝ

- <https://docs.microsoft.com/en-us/sql>
- <https://www.w3schools.com/sql>
- <https://www.sqlservertutorial.net/sql>
- [https://docs.oracle.com/cd/E17952\\_01/index.html](https://docs.oracle.com/cd/E17952_01/index.html)