**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

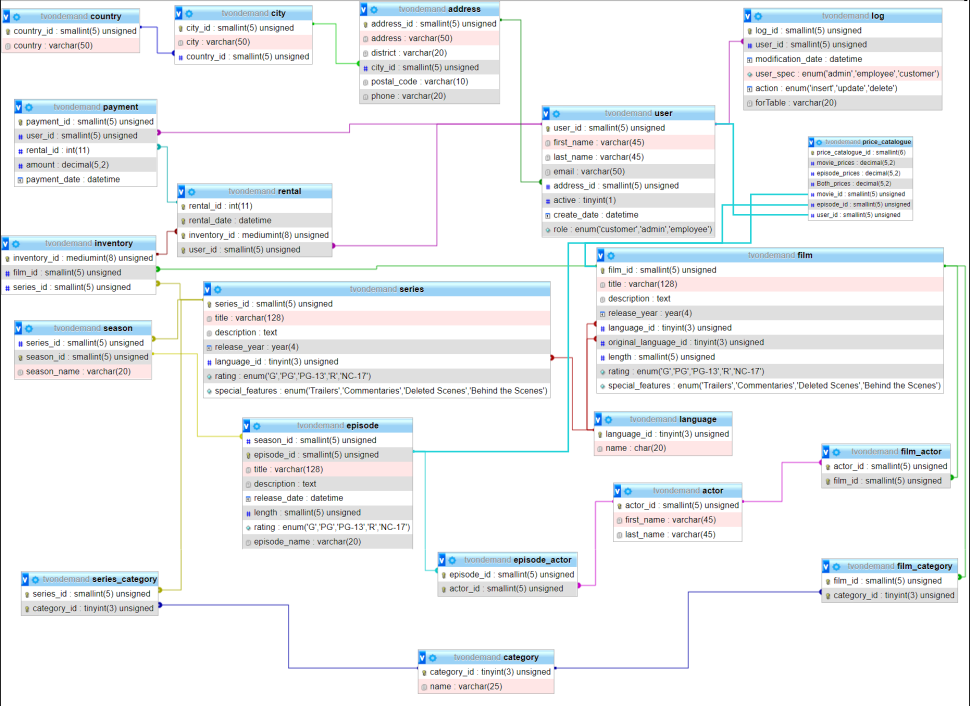
**ΚΕΦΑΛΑΙΟ 1**

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).



**Β)**

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 εγράφη 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 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

ΙNNER 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 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

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$$

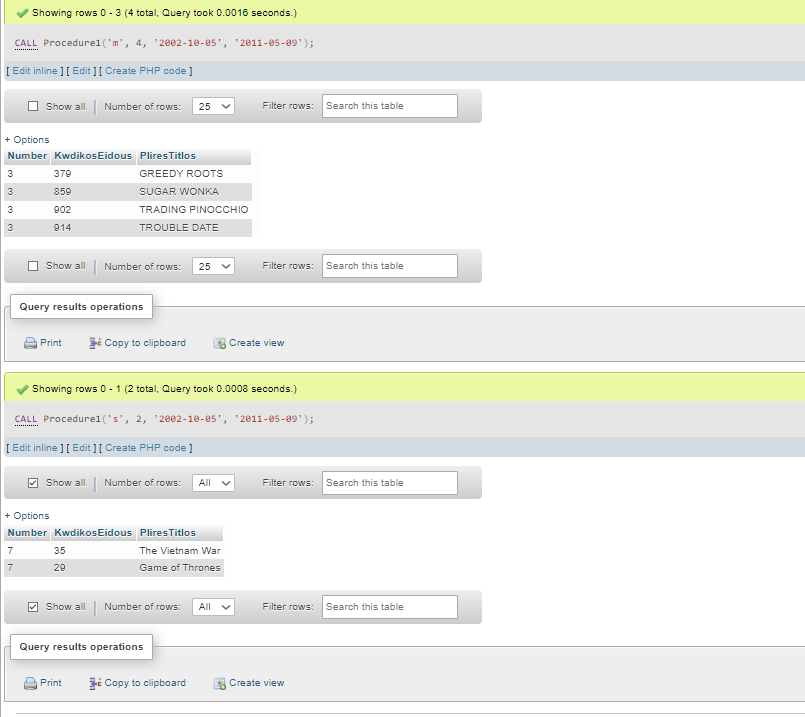
DELIMITER ;

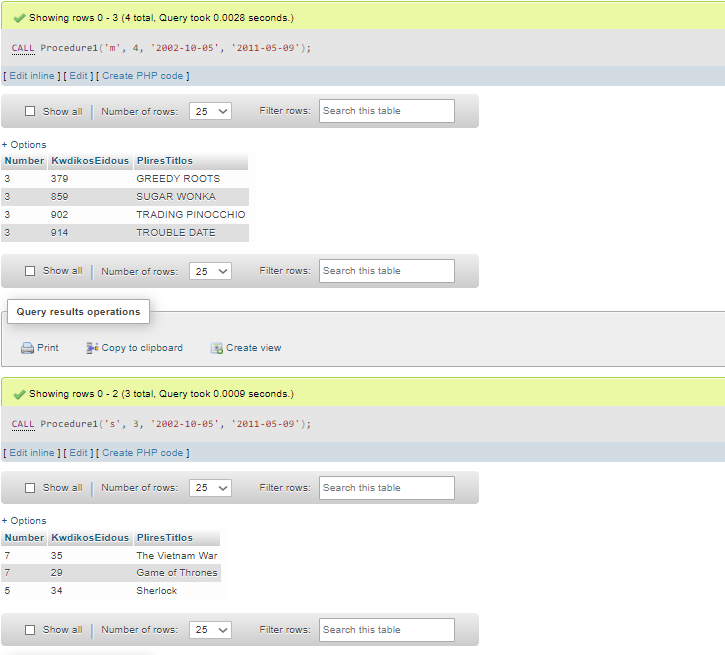
-- 1 result

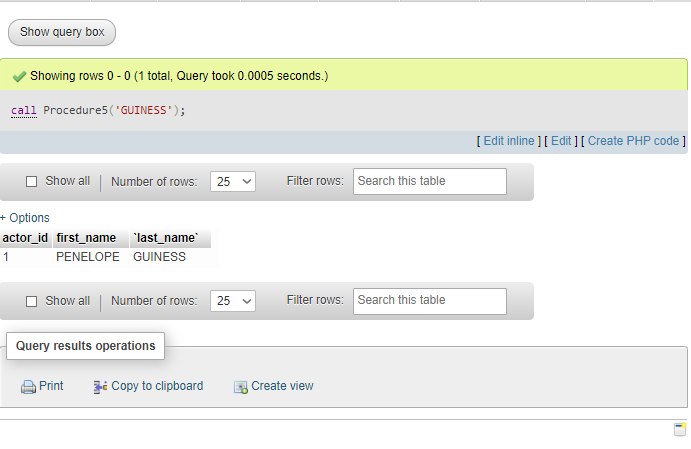
-- call Procedure4('GUINESS');

-- more than 1 results

-- call Procedure4('Livingston');







* Οι Stored Procedures δίνονται και στον φάκελο StoredProcedures σε .SQL αρχείο.

**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.w3schools.com/sql/default.asp)
* <https://www.sqlservertutorial.net/sql>
* <https://docs.oracle.com/cd/E17952_01/index.html>