**ΔΕΛΗΓΙΑΝΝΗ ΜΥΡΤΩ 1067389**

**ΝΙΚΟΛΟΥΔΗΣ ΠΑΝΑΓΙΩΤΗΣ 1067076**

**Project Βάσεων Δεδομένων**

**Παραδοχές**

Για να λειτουργήσουν οι δοθέντες προδιαγραφές, χρειάστηκε να γίνουν από εμάς οι παρακάτω παραδοχές:

* Ένας διευθυντής (manager) είναι επίσης υπάλληλος, ώστε η βάση να διαθέτει πληροφορίες σχετικά με το βιογραφικό του, τα πιστοποιητικά του, τις συστατικές του, τα βραβεία του και το επίπεδο της μόρφωσής του (πτυχία, μεταπτυχιακά κ.τ.λ.).
* Προσθέτουμε στη βάση τον πίνακα Evaluation\_parts, όπου θα διατίθεται πληροφορία για τα στάδια της κάθε αξιολόγησης, δηλαδή θα υπάρχουν πεδία για την αξιολόγηση της συνέντευξης ενός υπαλλήλου, για τα σχόλια της συνέντευξης, την βαθμολογία του υπαλλήλου από κάποιον διευθυντή, την αξιολόγηση των προσόντων του και το άθροισμα των παραπάνω επιμέρους αξιολογήσεων. Επίσης, είναι απαραίτητο να δημιουργηθούν 3 ακόμη πεδία, ένα που θα αποθηκεύει το username του υπαλλήλου που αξιολογείται, ένα αντίστοιχα για τον αξιολογητή, και ένα για το ID της θέσης εργασίας για την οποία αξιολογείται ο υπάλληλος. Τέλος, αφού ο συνδυασμός ID θέσης - υπαλλήλου θα είναι μοναδικός για κάθε αξιολόγηση, θα αποτελούν πρωτεύον κλειδί του πίνακα αυτού.
* Δημιουργούμε τον πίνακα Logs (πίνακας ενεργειών), ο οποίος, κάθε φορά που γίνεται μία ενέργεια εισαγωγής, επεξεργασίας ή διαγραφής δεδομένων στο σύστημα, καταγράφει στο πεδίο log την περιγραφή της ενέργειας αυτής.
* Σύμφωνα με τα ζητούμενα της εκφώνησης, πρέπει να δημιουργήσουμε άλλο ένα table, τον Administrator - διαχειριστή, ο οποίος είναι χρήστης, αλλά, σε αντίθεση με τους υπόλοιπους χρήστες, έχει το δικαίωμα να επεξεργαστεί δεδομένα στη βάση, όπως το προφίλ του και τα δεδομένα των αξιολογήσεων.

**Εισαγωγή Δεδομένων**

Αρχικά, δημιουργούμε 9 χρήστες, εκ των οποίων οι 5 είναι υπάλληλοι (3 απλοί υπάλληλοι, 2 διευθυντές), 2 αξιολογητές και 1 διαχειριστή. Οι εταιρείες που δημιουργήσαμε είναι 2, διαφορετικής φύσεως, με 2 θέσεις εργασίας για την πρώτη και 1 για την δεύτερη και τα αντικείμενα με τα οποία ασχολούνται. Έπειτα εισάγουμε κάποια αιτήματα αξιολογήσεων για τις συγκεκριμένες θέσεις εργασίας που έχουν υποβάλλει οι υπάλληλοι, τις επιμέρους αξιολογήσεις τους από τους αξιολογητές και, τέλος, τα αποτελέσματά τους. Στον πίνακα πτυχία (Degree) προσθέσαμε 10 διαφορετικά μεταξύ τους πτυχία, τα οποία αποδίδονται σε υπαλλήλους και διευθυντές μέσω του πίνακα Has\_Degree. Τέλος, στον πίνακα Languages συνδέσαμε κάθε υπάλληλο ή διευθυντή με μία ή περισσότερες γλώσσες τις οποίες αυτός γνωρίζει.

**Stored Procedures**

* Αρχικά, δημιουργούμε ένα stored procedure, το οποίο δέχεται ως είσοδο το όνομα ενός υπαλλήλου και ενός αξιολογητή, υπολογίζει το άθροισμα των επιμέρους αξιολογήσεων που έχουν πραγματοποιηθεί από τον αξιολογητή για τον υπάλληλο και το αποθηκεύει στο πεδίο sum\_grade του πίνακα Evaluation\_parts. Αυτή η διαδικασία λειτουργεί μέσω του trigger sum\_grade\_trig, το οποίο, μετά από κάθε εισαγωγή στον πίνακα Evaluation\_parts την ενεργοποιεί ώστε να εισαχθεί και στο πεδίο sum\_grade το άθροισμα των επιμέρους αξιολογήσεων.
* Ακολουθεί η διαδικασία getEmployeeEvv, η οποία δέχεται το όνομα και το επώνυμο ενός υπαλλήλου και τυπώνει όλα τα αιτήματα αξιολόγησης που έχει υποβάλλει ο υπάλληλος, τα αποτελέσματα των ολοκληρωμένων αξιολογήσεων και το όνομα του αξιολογητή που αντιστοιχεί σε αυτές.
* Έπειτα, το procedure updateResult, που παίρνει ως είσοδο τον κωδικό μιας θέσης προαγωγής και το username ενός αξιολογητή, ελέγχει αν υπάρχουν ολοκληρωμένες αξιολογήσεις που δεν έχουν καταχωρηθεί ως οριστικές και μεταφέρει τον συνολικό βαθμό της αξιολόγησης στον πίνακα Evaluation\_results, όπου θα βρίσκονται οι τελικοί συνολικού βαθμοί όλων των οριστικοποιημένων αξιολογήσεων.
* Τέλος, δημιουργήσαμε τη διαδικασία getEvStatus, η οποία δέχεται ως είσοδο τον κωδικό μιας θέσης προαγωγής. Τυπώνει το μήνυμα «Οριστικοποιημένοι Πίνακες: » και τις ολοκληρωμένες αξιολογήσεις για τη συγκεκριμένη θέση εργασίας με φθίνουσα σειρά βάσει του τελικού βαθμού και το μήνυμα «Αξιολόγηση σε εξέλιξη… Εκκρεμούν: » μαζί με τον αριθμό των αιτημάτων που εκκρεμούν.

**Triggers:**

* Αρχικά έχουμε κατασκευάσει έναν trigger (τον sum\_grade\_trig) ο οποίος όπως προαναφέρουμε σε συνδυασμό με την stored procedure “resultSum” που είναι εμφωλευμένη στο σώμα του,ουσιαστικά γεμίζει το πεδίο sum\_grade του πίνακα evaluation\_parts. Πιο αναλυτικά,ο trigger είναι προγραμματισμένος με το που γίνεται εγγραφή στον πίνακα evaluation\_parts, να καλεί την “resultSum” (βλ. Stored Procedures).
* Έπειτα πρέπει μα κατασκευάσουμε triggers οι οποίοι ενημερώνουν τον πίνακα logs, ανάλογα με το αν γίνεται εγγραφή, διαγράφη ή ακόμα και επεξεργασία στους πίνακες Job,Employee και Evaluation\_Request. Για τον κάθε έναν από τους παραπάνω πίνακες θα κατασκευάσουμε 3 triggers εφόσον υπάρχουν 3 δυνατές ενέργειες(insert,delete,update) που μπορούν να υποστούν.
* Για τον πίνακα **Job** λοιπόν, κατασκευάζουμε τους:

1. **Job\_Insert**: Αφού γίνει insert στον πίνακα job , αναγνωρίζει το id της θέσης εργασίας που μόλις προστέθηκε(μέσω της εντολής SELECT INTO) και το τυπώνει-εγγράφει , μαζί με ένα φιλικό προς το χρήστη μήνυμα, στο πεδίο log(TEXT) του πίνακα Logs.
2. **Job\_Delete**: Προτού γίνει delete στον πίνακα job , αναγνωρίζει το id της θέσης εργασίας που επρόκειτο να διαγραφεί (μέσω της εντολής SELECT INTO) και το τυπώνει-εγγραφεί , μαζί με ένα φιλικό προς το χρήστη μήνυμα, στο πεδίο log(TEXT) του πίνακα Logs.
3. **Job\_Update:** Αφού γίνει update στον πίνακα job , ελέγχει αν το id θέσης εργασίας παρέμεινε ίδιο. Αν είναι διαφορετικό τότε αναγνωρίζει το id που μόλις προστέθηκε και το τυπώνει- εγγράφει καθώς και το παλιό id , μαζί με ένα φιλικό προς το χρήστη μήνυμα, στο πεδίο log(TEXT) του πίνακα Logs. Αν το id δεν άλλαξε απλα ενημερώνει τον χρήστη με ένα αντίστοιχο μήνυμα(στον πίνακα Logs) πως υπάρχει κάποια αλλαγή σε κάποιο από τα πεδία της θέσης εργασίας με id=με το id που εχει αναγνωριστεί από τις εντολες SELECT INTO.

* Για τον πίνακα **Employee**, κατασκευάζουμε τους:

1. **Employee\_Insert**: Αφού γίνει insert στον πίνακα employee, αναγνωρίζει το username του υπαλλήλου που μόλις προστέθηκε(μέσω της εντολής SELECT INTO) και το τυπώνει-εγγραφει , μαζί με ένα φιλικό προς το χρήστη μήνυμα, στο πεδίο log(TEXT) του πίνακα Logs.
2. **Employee\_Delete**: Προτού γίνει delete στον πίνακα employee , αναγνωρίζει το username του υπαλλήλου που επρόκειτο να διαγραφεί (μέσω της εντολής SELECT INTO) και το τυπώνει-εγγραφεί , μαζί με ένα φιλικό προς το χρήστη μήνυμα, στο πεδίο log(TEXT) του πίνακα Logs.
3. **Employee\_Update:** Αφού γίνει update στον πίνακα employee, ελέγχει αν το username του υπαλλήλου παρέμεινε ίδιο. Αν είναι διαφορετικό τότε αναγνωρίζει το username που μόλις προστέθηκε και το τυπώνει-εγγραφει καθώς και το παλιό username , μαζί με ένα φιλικό προς το χρήστη μήνυμα, στο πεδίο log(TEXT) του πίνακα Logs. Αν το username δεν άλλαξε απλα ενημερώνει τον χρήστη με ένα αντίστοιχο μήνυμα(στον πίνακα Logs) πως υπάρχει κάποια αλλαγή σε κάποιο από τα πεδία του υπαλλήλου με username=με το username που εχει αναγνωριστεί από τις εντολες SELECT INTO.

* Για τον πίνακα **Evaluation\_Request**, κατασκευάζουμε τους:

1. **Ev\_Request\_Insert**: Αφού γίνει insert στον πίνακα Evaluation\_Request, αναγνωρίζει το username του υπαλλήλου και το id της θέσης εργασίας του αιτήματος που προστέθηκε(μέσω της εντολής SELECT INTO) και τα τυπώνει-εγγραφει , μαζί με ένα φιλικό προς το χρήστη μήνυμα, στο πεδίο log(TEXT) του πίνακα Logs.
2. **Εv\_Request\_Delete:** Προτού γίνει delete στον πίνακα employee , αναγνωρίζει το username του υπαλλήλου και το id της θέσης εργασίας του αιτήματος που επρόκειτο να διαγραφεί (μέσω της εντολής SELECT INTO) και το τυπώνει-εγγραφεί , μαζί με ένα φιλικό προς το χρήστη μήνυμα, στο πεδίο log(TEXT) του πίνακα Logs.
3. **Ev\_Request\_Update:** Αφού γίνει update στον πίνακα employee, ελέγχει αν το username του υπαλλήλου παρέμεινε ίδιο. Αν είναι διαφορετικό τότε αναγνωρίζει το username του εργαζομένου που μόλις εκανε αλλαγή στην αιτηση, και το τυπώνει-εγγραφει, καθώς και το παλιό username , μαζί με ένα φιλικό προς το χρήστη μήνυμα, στο πεδίο log(TEXT) του πίνακα Logs. Αν το username δεν άλλαξε απλα ενημερώνει τον χρήστη με ένα αντίστοιχο μήνυμα(στον πίνακα Logs) πως υπάρχει κάποια αλλαγή σε κάποιο από τα πεδία της αίτησης που έκανε ο εργαζόμενος με username=με το username που εχει αναγνωριστεί από τις εντολες SELECT INTO.

* Στην συνέχεια η εκφώνηση μας ζητά να κατασκευάσουμε έναν trigger , ο οποίος πρακτικά να απαγορεύει την οποιαδήποτε αλλαγή στα πεδια ΑΦΜ,ΔΟΥ και Όνομα του πίνακα **Company**. Για να το πετύχουμε αυτό, θα κατασκευάσουμε τον trigger:
* **Company\_Update:** Στην περίπτωση που γινει update στον πίνακα company, ελεγχει αν οι τιμές των AFM,DOY και Name (που αναγνωρίστηκαν μεσω των εντολών SELECT INTO) έχουν ανανεωθεί. Αν οι τιμές είναι πράγματι διαφορετικές, τότε με την εντολή **SET** τους ανατίθεται η παλιά τους τιμή ενώ τυπώνει στον χρήστη και ένα μήνυμα στο οποίο τον ενημερώνει πως το ΑΦΜ,η ΔΟΥ και το Όνομα θα διατηρήσουν τις τιμές τους.
* Τελος, ζητείται από την εκφώνηση να κατασκευάσουμε ένα trigger το οποίο ουσιαστικά να εξετάζει αν ο χρήστης που κάνει την οποιαδήποτε αλλαγή ή εγγραφή ή διαγραφή,είναι administrator.Στην περίπτωση που δεν είναι administrator ομώς να του απαγορεύει τις παραπάνω ενέργειες.

Εφόσον δεν είχαμε χρόνο όμως να ασχολήθουμε με το δεύτερο κομματι της εργασίας δηλαδη το GUI ο συγκεκριμένος trigger δεν μπορεί να υλοποιηθεί καθώς δεν υπάρχει η οποιαδήποτε “log-in” διαδικασία σε αυτή τη φάση του project.

**Κώδικας:**

DROP DATABASE IF EXISTS Evaluation\_system;

CREATE DATABASE Evaluation\_system;

USE Evaluation\_system;

CREATE TABLE User(

Username VARCHAR(12) DEFAULT 'unknown' NOT NULL,

Password VARCHAR(10) DEFAULT 'unknown' NOT NULL,

Name VARCHAR(25) DEFAULT 'unknown' NOT NULL,

Surname VARCHAR(25) DEFAULT 'unknown' NOT NULL,

reg\_date DATETIME NOT NULL,

email VARCHAR(30) DEFAULT 'unknown' NOT NULL,

PRIMARY KEY(Username)

);

CREATE TABLE Administrator(

Admin\_username VARCHAR(12) DEFAULT 'unknown' NOT NULL,

PRIMARY KEY(Admin\_username),

CONSTRAINT ADMINUSERNAME

FOREIGN KEY (Admin\_username) REFERENCES User(Username)

ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE Company(

AFM CHAR(9) NOT NULL,

DOY CHAR(15) NOT NULL,

Name VARCHAR(35) DEFAULT 'unknown' NOT NULL,

Phone BIGINT(16) NOT NULL,

street VARCHAR(15) DEFAULT 'unknown' NOT NULL,

num TINYINT(6) NOT NULL,

city VARCHAR(15) DEFAULT 'unknown' NOT NULL,

country VARCHAR(15) DEFAULT 'unknown' NOT NULL,

PRIMARY KEY (AFM)

);

CREATE TABLE Employee(

Emp\_Username VARCHAR(12) DEFAULT 'unknown' NOT NULL,

Bio TEXT(255),

sistatikes VARCHAR(35),

certificates VARCHAR(35),

awards VARCHAR(35),

PRIMARY KEY(Emp\_Username),

CONSTRAINT EMPUSERNAME

FOREIGN KEY (Emp\_Username) REFERENCES User(Username)

ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE Manager(

Man\_Username VARCHAR(12) DEFAULT 'unknown' NOT NULL,

Exp\_years TINYINT(4) NOT NULL,

Firm CHAR(9) DEFAULT 'unknown' NOT NULL,

PRIMARY KEY(Man\_Username),

CONSTRAINT MANUSERNAME

FOREIGN KEY (Man\_Username) REFERENCES Employee(Emp\_Username)

ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (Firm) REFERENCES Company(AFM)

ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE Evaluator(

Evl\_Username VARCHAR(12) DEFAULT 'unknown' NOT NULL,

Years\_exp TINYINT(4) NOT NULL,

Cmpny CHAR(9) DEFAULT 'unknown' NOT NULL,

PRIMARY KEY(Evl\_Username),

CONSTRAINT EVLUSERNAME

FOREIGN KEY (Evl\_Username) REFERENCES User(Username)

ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY(Cmpny) REFERENCES Company(AFM)

ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE Job(

Id INT(4) NOT NULL AUTO\_INCREMENT,

Start\_date DATE NOT NULL,

Salary FLOAT(6) NOT NULL,

Position VARCHAR(40) DEFAULT 'unknown' NOT NULL,

Edra VARCHAR(45) DEFAULT 'unknown' NOT NULL,

evaltr VARCHAR(12) DEFAULT 'unknown' NOT NULL,

announce\_date DATETIME NOT NULL,

submission\_date DATE NOT NULL,

PRIMARY KEY(Id),

UNIQUE(evaltr),

CONSTRAINT EVALUATION

FOREIGN KEY (evaltr) REFERENCES Evaluator(Evl\_Username)

ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE Job\_field(

field\_title VARCHAR(36) DEFAULT 'unknown' NOT NULL,

field\_description TINYTEXT,

belongs\_to VARCHAR(36) DEFAULT 'unknown',

PRIMARY KEY (field\_title),

CONSTRAINT BELONGS

FOREIGN KEY (belongs\_to) REFERENCES Job\_field(field\_title)

ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE Job\_needs(

Id\_Job INT(4) NOT NULL,

title\_field VARCHAR(36) DEFAULT 'unknown' NOT NULL,

PRIMARY KEY (Id\_Job),

CONSTRAINT IDJOB

FOREIGN KEY (Id\_Job) REFERENCES Job(Id)

ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT TITLEFIELD

FOREIGN KEY (title\_field) REFERENCES Job\_field(field\_title)

ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE Evaluation\_Request(

empl\_username VARCHAR(12) DEFAULT 'unknown' NOT NULL,

JobID INT(4) NOT NULL,

PRIMARY KEY (empl\_username,JobID),

CONSTRAINT EMPLNAME

FOREIGN KEY (empl\_username) REFERENCES Employee(Emp\_Username)

ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT JOBIDENTIFICATION

FOREIGN KEY (JobID) REFERENCES Job(Id)

ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE Evaluation\_parts(

empl VARCHAR(12) DEFAULT 'unknown' NOT NULL,

JobIdent INT(4) NOT NULL,

evltr VARCHAR(12) DEFAULT 'unknown' NOT NULL,

interview\_grade REAL CHECK (interview\_grade>0 AND interview\_grade<4),

interview\_comments VARCHAR(255) DEFAULT 'no comments' NOT NULL,

manager\_report\_ev REAL CHECK (manager\_report\_ev>0 AND manager\_report\_ev<4),

qualifications\_ev REAL CHECK (qualifications\_ev>0 AND qualifications\_ev<2),

sum\_grade REAL CHECK (sum\_grade>0 AND sum\_grade<10),

PRIMARY KEY(empl,evltr),

UNIQUE (sum\_grade,interview\_comments),

CONSTRAINT EMPLO

FOREIGN KEY (empl,JobIdent) REFERENCES Evaluation\_Request(empl\_username,JobID)

ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT EVLTR

FOREIGN KEY (evltr) REFERENCES Job(evaltr)

ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE Evaluation\_results(

Ev\_Id INT(4) NOT NULL AUTO\_INCREMENT,

em\_username VARCHAR(12) DEFAULT 'unknown' NOT NULL,

Job\_Id INT(4) NOT NULL,

Grade REAL NOT NULL,

Comments VARCHAR(255) DEFAULT 'no comments' NOT NULL,

PRIMARY KEY(Ev\_Id),

CONSTRAINT GRADEANDCOMMS

FOREIGN KEY (Grade,Comments) REFERENCES Evaluation\_parts(sum\_grade,interview\_comments)

ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT JOBIDENT

FOREIGN KEY (em\_username,Job\_Id) REFERENCES Evaluation\_Request(empl\_username,JobID)

ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE Degree(

degree\_title VARCHAR(50) DEFAULT 'unknown' NOT NULL,

institute VARCHAR(40) DEFAULT 'unknown' NOT NULL,

education\_level ENUM('LYKEIO','UNIVERSITY','MASTER','PHD'),

PRIMARY KEY(degree\_title,institute)

);

CREATE TABLE Languages(

username\_em VARCHAR(12) DEFAULT 'unknown' NOT NULL,

language SET('EN','FR','SP','GR'),

CONSTRAINT USERNAME

FOREIGN KEY (username\_em) REFERENCES Employee(Emp\_Username)

ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE Has\_Degree(

username\_emp VARCHAR(12) DEFAULT 'unknown' NOT NULL,

deg\_title VARCHAR(50) DEFAULT 'unknown' NOT NULL,

inst VARCHAR(40) DEFAULT 'unknown' NOT NULL,

graduation\_year YEAR(4),

d\_grade FLOAT(3),

PRIMARY KEY(username\_emp,deg\_title),

CONSTRAINT USERNAMEEMP

FOREIGN KEY (username\_emp) REFERENCES Employee(Emp\_Username)

ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT DEGTITLEANDINSTIT

FOREIGN KEY (deg\_title,inst) REFERENCES Degree(degree\_title,institute)

ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE Logs(

log\_num INT AUTO\_INCREMENT NOT NULL,

log TEXT,

PRIMARY KEY (log\_num)

);

INSERT INTO User VALUES

('panic', '87654', 'Panagiotis', 'Nikolaidis', '2014-04-01 00:00:00 PM', 'panic@gmail.com'),

('kpap', '12345', 'Kostas', 'Papageorgiou', '2019-09-17 00:00:00 PM', 'kpap@gmail.com'),

('maalex', '23456', 'Maria', 'Alexopoulou', '2017-05-30 00:00:00 PM', 'maalex@gmail.com'),

('anamakri', '34567', 'Anastasia', 'Makri', '2014-07-12 00:00:00 PM', 'anamakri@gmail.com'),

('sdem', '45678', 'Stavros', 'Demetrakopoulos', '2020-10-05 00:00:00 PM', 'sdem@gmail.com'),

('dimanagno', '56789', 'Dimitris', 'Anagnostou', '2015-03-08 00:00:00 PM', 'dimanagno@gmail.com'),

('jlazari', '01234', 'Ioanna', 'Lazari', '2018-12-03 00:00:00 PM', 'jlazari@gmail.com'),

('vpapad', '09876', 'Vassilis', 'Papadopoulos', '2016-01-04 00:00:00 PM', 'vpapad@gmail.com'),

('annadiama', '98765', 'Anna', 'Diamantopoulou', '2019-04-03 00:00:00 PM)', 'annadiama@gmail.com');

INSERT INTO Company VALUES

('556783340', 'A PATRWN', 'Drymot', '2610743892', 'Maizonos', 125, 'Patras', 'Greece'),

('875835632', 'B ATHINWN', 'Telem', '2108342954', 'Kanakari', 204, 'Athens', 'Greece');

INSERT INTO Administrator VALUES

('panic');

INSERT INTO Employee VALUES

('kpap', 'Software Engineer', 'Sistatiki', 'Certificate', 'Aw1'),

('anamakri', 'Oinologos, Xhmikos, eidikeush stin texnologia trofimwn, Dieuthuntria Etaireias Potopoieias Drymot', 'Certificate', 'Sistatiki', 'Epixeirhmatias ths xronias'),

('dimanagno', 'Hlektrologos Mhxanikos kai Mhxanikos H/Y, Dieuthunths ths Telem', 'Certificate', 'Sistatiki', NULL),

('vpapad', 'Xhmikos, eidikeush sth texnologia trofimwn', 'Sistatiki', 'Certificate', 'Aw2'),

('annadiama', 'Empeiria ws grammateas', 'Sistatiki', 'Certificate', 'Ypallhlos tou mhna');

INSERT INTO Manager VALUES

('anamakri', 10, '556783340'),

('dimanagno', 15, '875835632');

INSERT INTO Evaluator VALUES

('maalex', 8, '556783340'),

('sdem', 4, '556783340'),

('jlazari', 6, '875835632');

INSERT INTO Job VALUES

(NULL, '2021-12-03', 1500.00, 'Chemist', 'Drymot', 'maalex', '2020-11-30', '2020-12-17'),

(NULL, '2021-02-05', 1000.00, 'Oinologos','Drymot', 'sdem', '2020-12-13', '2021-01-03'),

(NULL, '2021-03-07', 1600.00, 'Web Developer', 'Telem', 'jlazari', '2020-12-05', '2021-01-05');

INSERT INTO Job\_field VALUES

('Beverage quality check','Test the quality of the products of the company'),

('Web Developer', 'Create and check the website of a company');

INSERT INTO Job\_needs VALUES

('1', 'Beverage quality check'),

('2', 'Web Developer');

INSERT INTO Evaluation\_Request VALUES

('anamakri', 'Oinologos'),

('kpap', 'Web Developer');

INSERT INTO Evaluation\_parts VALUES

('anamakri', 2, 'sdem', 2, 'Agxwmenh, arketa kleisth, wstoso edwse swstes apanthseis', 3, 4, NULL),

('kpap', 3, 'jlazari', 3, 'Koinwnikos, me autopepoithisi, kales apanthseis', 3, 4, NULL);

INSERT INTO Degree VALUES

('Computer Science', 'University of Ioannina', 'UNIVERSITY'),

('Computer Security', 'University of Aegean', 'MASTER'),

('Chemistry', 'University of Athens', 'UNIVERSITY'),

('Oenology', 'University of W.Attica', 'UNIVERSITY'),

('Food Technology', 'University of Peloponnese', 'UNIVERSITY'),

('Chemistry', 'University of Thessaloniki', 'MASTER'),

('Electrical and Computer Science', 'University of Patras', 'UNIVERSITY'),

('Computer Engineer', 'University of Patras', 'MASTER'),

('Food Technology', 'University of Thessaloniki', 'UNIVERSITY'),

('Computer Science', 'University of Thessaly', 'UNIVERSITY');

INSERT INTO Languages VALUES

('kpap', 'EN' 'FR' 'GR'),

('anamakri', 'EN' 'FR' 'GR'),

('dimanagno', 'EN' 'SP' 'GR'),

('vpapad', 'EN' 'GR'),

('annadiama', 'EN' 'FR' 'GR');

INSERT INTO Has\_Degree VALUES

('kpap', 'Computer Science', 'University of Ioannina', 2005, 8.5),

('kpap', 'Computer Security', 'University of Aegean', 2008, 9),

('anamakri', 'Oenology', 'University of W.Attica', 2000, 8),

('anamakri', 'Food Technology', 'University of Peloponnese', 2005, 9),

('anamakri', 'Chemistry', 'University of Thessaloniki', 2009, 7.5),

('dimanagno', 'Electrical and Computer Science', 'University of Patras', 2006, 7),

('dimanagno', 'Computer Engineer', 'University of Patras', 2010, 8.5),

('vpapad', 'Food Technology', 'University of Thessaloniki', 2004, 8),

('vpapad', 'Chemistry', 'University of Thessaloniki', 2007, 6.5);

DROP PROCEDURE IF EXISTS resultSum;

DELIMITER $

CREATE PROCEDURE resultSum(empuname VARCHAR(12), evluname VARCHAR(12))

BEGIN

SELECT SUM(interview\_grade + manager\_report\_ev + qualifications\_ev) INTO sum\_grade FROM Evaluation\_parts

WHERE empl = empuname AND evltr = evluname;

END $

DELIMITER;

DROP TRIGGER IF EXISTS sum\_grade\_trig;

DELIMITER $

CREATE TRIGGER sum\_grade\_trig

AFTER INSERT ON Evaluation\_parts

FOR EACH ROW

BEGIN

DECLARE username\_of\_emp VARCHAR(12);

DECLARE username\_of\_evl VARCHAR(12);

SELECT empl INTO username\_of\_emp FROM Evaluation\_parts;

SELECT evltr INTO username\_of\_evl FROM Evaluation\_parts;

CALL resultSum(username\_of\_emp,username\_of\_evl);

END $

DELIMITER ;

DROP PROCEDURE IF EXISTS getEmployeeEvv;

DELIMITER $

CREATE PROCEDURE getEmployeeEvv(ename VARCHAR(25), esurname VARCHAR(25))

BEGIN

SELECT \* FROM Evaluation\_Request

INNER JOIN Employee ON Emp\_Username = empl\_username

INNER JOIN User ON Username = Emp\_Username

WHERE Name = ename AND Surname = esurname;

SELECT \* FROM Evaluation\_results

INNER JOIN Employee ON Emp\_username = em\_username

INNER JOIN User ON Username = Emp\_Username

WHERE Name = ename AND Surname = esurname;

SELECT Name, Surname FROM User

INNER JOIN Employee ON Emp\_Username = Username

INNER JOIN Evaluator ON Evl\_Username = Username

INNER JOIN Evaluation\_parts ON evltr = Evl\_Username

WHERE Name = ename AND Surname = esurname;

END $

DELIMITER;

DROP PROCEDURE IF EXISTS updateResult;

DELIMITER $

CREATE PROCEDURE updateResult(jid INT(4), evuname VARCHAR(12))

BEGIN

DECLARE summ REAL;

DECLARE not\_found INT;

DECLARE ucursor CURSOR FOR

SELECT sum\_grade FROM Evaluation\_parts

WHERE evltr = evuname;

DECLARE CONTINUE HANDLER FOR NOT FOUND

SET not\_found = 1;

SET not\_found = 0;

OPEN ucursor;

REPEAT

FETCH ucursor INTO summ;

IF(not\_found = 0)

THEN

UPDATE Evaluation\_results

SET Grade = summ

WHERE ev\_username = evuname AND Job\_Id = jid;

END IF;

UNTIL(not\_found = 1)

END REPEAT;

CLOSE ucursor;

END $

DELIMITER;

DROP PROCEDURE IF EXISTS getEvStatus;

DELIMITER $

CREATE PROCEDURE getEvStatus(jjid INT(4))

BEGIN

DECLARE emuname VARCHAR(12);

DECLARE not\_found INT;

DECLARE scursor CURSOR FOR

SELECT empl\_username FROM Evaluation\_Request

WHERE Job\_Id = jjid;

DECLARE CONTINUE HANDLER FOR NOT FOUND

SET not\_found = 1;

SET not\_found = 0;

OPEN scursor;

REPEAT

FETCH scursor INTO emuname;

IF(not\_found = 0)

THEN

PRINT "Oristikopoihmenoi pinakes:";

SELECT \* FROM Evaluation\_results

WHERE Job\_Id = jjid AND em\_username = emuname

ORDER BY Grade DESC;

PRINT "Aksiologisi se ekseliksi...";

PRINT "Ekremmoun:";

SELECT count(\*) FROM Evaluation\_Request

WHERE JobID = jjid AND empl\_username = emuname;

END IF;

UNTIL(not\_found = 1)

END REPEAT;

CLOSE scursor;

END $

DELIMITER;

DROP TRIGGER IF EXISTS Job\_Insert;

DELIMITER $

CREATE TRIGGER Job\_Insert

AFTER INSERT ON Job

FOR EACH ROW

BEGIN

DECLARE id\_of\_job INT(4);

SELECT MAX(Id) INTO id\_of\_job FROM Job;

INSERT INTO Logs(log)

VALUES('A Job with the id of '+ @id\_of\_job +' was inserted in the table "Job".');

END $

DELIMITER ;

DROP TRIGGER IF EXISTS Job\_Delete;

DELIMITER $

CREATE TRIGGER Job\_Delete

BEFORE DELETE ON Job

FOR EACH ROW

BEGIN

DECLARE id\_of\_job INT(4);

SELECT Id INTO id\_of\_job FROM Job;

INSERT INTO Logs(log)

VALUES('A Job with the id of '+ @id\_of\_job +' was deleted by the table "Job".');

END $

DELIMITER ;

DROP TRIGGER IF EXISTS Job\_Update;

DELIMITER $

CREATE TRIGGER Job\_Update

AFTER UPDATE ON Job

FOR EACH ROW

BEGIN

DECLARE id\_of\_job INT(4);

SELECT Id INTO id\_of\_job FROM Job;

IF (OLD.id\_of\_job<>NEW.id\_of\_job) THEN

INSERT INTO Logs(log)

VALUES ('The Job with the id of '+@OLD.id\_of\_job+' has now the id of '+@NEW.id\_of\_job+' . There might be some other changes too.');

ELSE

INSERT INTO Logs(log)

VALUES('A Job with the id of '+ @id\_of\_job +' was updated in the table "Job".');

END IF;

END $

DELIMITER ;

DROP TRIGGER IF EXISTS Employee\_Insert;

DELIMITER $

CREATE TRIGGER Employee\_Insert

AFTER INSERT ON Employee

FOR EACH ROW

BEGIN

DECLARE username\_of\_emp VARCHAR(12);

SELECT Emp\_Username INTO username\_of\_emp FROM Employee;

INSERT INTO Logs(log)

VALUES ('An Employee with the username of '+@username\_of\_emp+' has been inserted in the "Employee" table.');

END $

DELIMITER ;

DROP TRIGGER IF EXISTS Employee\_Delete;

DELIMITER $

CREATE TRIGGER Employee\_Delete

BEFORE DELETE ON Employee

FOR EACH ROW

BEGIN

DECLARE username\_of\_emp VARCHAR(12);

SELECT Emp\_Username INTO username\_of\_emp FROM Employee;

INSERT INTO Logs(log)

VALUES ('The Employee with the username of '+@username\_of\_emp+' has been deleted from the "Employee" table.');

END $

DELIMITER ;

DROP TRIGGER IF EXISTS Employee\_Update;

DELIMITER $

CREATE TRIGGER Employee\_Update

AFTER UPDATE ON Job

FOR EACH ROW

BEGIN

DECLARE username\_of\_emp VARCHAR(12);

SELECT Id INTO username\_of\_emp FROM Job;

IF (OLD.username\_of\_emp=NEW.username\_of\_emp) THEN

INSERT INTO Logs(log)

VALUES('An Employee with the username of '+ @username\_of\_emp +' was updated in the table "Employee".');

ELSE

INSERT INTO Logs(log)

VALUES ('The Employee with the username of '+@OLD.username\_of\_emp+' has now the username '+@NEW.username\_of\_emp+' . There might be some other changes too.');

END IF;

END $

DELIMITER ;

DROP TRIGGER IF EXISTS Ev\_Request\_Insert;

DELIMITER $

CREATE TRIGGER Ev\_Request\_Insert

AFTER INSERT ON Evaluation\_Request

FOR EACH ROW

BEGIN

DECLARE username\_of\_emp VARCHAR(12);

DECLARE id\_of\_job INT(4);

SELECT empl\_username INTO username\_of\_emp FROM Evaluation\_Request;

SELECT JobID INTO username\_of\_emp FROM Evaluation\_Request;

INSERT INTO Logs(log)

VALUES ('An Employee with the username of '+@username\_of\_emp+' has requested evaluation for the Job with the Id of '+@id\_of\_job+' .');

END $

DELIMITER ;

DROP TRIGGER IF EXISTS Ev\_Request\_Delete;

DELIMITER $

CREATE TRIGGER Ev\_Request\_Delete

BEFORE DELETE ON Evaluation\_Request

FOR EACH ROW

BEGIN

DECLARE username\_of\_emp VARCHAR(12);

SELECT empl\_username INTO username\_of\_emp FROM Evaluation\_Request;

INSERT INTO Logs(log)

VALUES ('The Employee with the username of '+@username\_of\_emp+' has been deleted from the "Evaluation\_Request" table.');

END $

DELIMITER ;

DROP TRIGGER IF EXISTS Ev\_Request\_Update;

DELIMITER $

CREATE TRIGGER Ev\_Request\_Update

AFTER UPDATE ON Evaluation\_Request

FOR EACH ROW

BEGIN

DECLARE username\_of\_emp VARCHAR(12);

SELECT Id INTO username\_of\_emp FROM Evaluation\_Request;

IF (OLD.username\_of\_emp=NEW.username\_of\_emp) THEN

INSERT INTO Logs(log)

VALUES('An Employee with the username of '+ @username\_of\_emp +' was updated in the table "Evaluation\_Request".');

ELSE

INSERT INTO Logs(log)

VALUES ('The Employee with the username of '+@OLD.username\_of\_emp+' has now the username of '+@NEW.username\_of\_emp+' . There might be some other changes in the "Evaluation\_Request" table too.');

END IF;

END $

DELIMITER ;

DROP TRIGGER IF EXISTS Company\_Update;

DELIMITER $

CREATE TRIGGER Company\_Update

AFTER UPDATE ON Company

FOR EACH ROW

BEGIN

DECLARE comp\_afm CHAR(9);

DECLARE comp\_doy CHAR(15);

DECLARE comp\_name VARCHAR(35);

SELECT AFM INTO comp\_afm FROM Company;

SELECT DOY INTO comp\_doy FROM Company;

SELECT Name INTO comp\_name FROM Company;

IF (OLD.comp\_afm<>NEW.comp\_afm || OLD.comp\_doy<>NEW.comp\_doy || OLD.comp\_name<>NEW.comp\_name) THEN

SET OLD.comp\_afm=NEW.comp\_afm;

SET OLD.comp\_doy=NEW.comp\_doy;

SET OLD.comp\_name=NEW.comp\_name;

PRINT 'WARNING: AFM,DOY and Company Name will detain their values.';

END IF;

END $

DELIMITER ;

DROP TRIGGER IF EXISTS Admin\_check;

DELIMITER $

CREATE TRIGGER Admin\_check

BEFORE INSERT ON User

BEGIN

FOR EACH ROW

END$

DELIMITER ;