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. **Ev_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.

<u>Κώδικας:</u>

```
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 ;
```

//den leitourgei

```
DROP TRIGGER IF EXISTS Admin_check;
DELIMITER $
CREATE TRIGGER Admin_check
BEFORE INSERT ON User
BEGIN
FOR EACH ROW

END$
DELIMITER ;
```