



CEID
COMPUTER ENGINEERING & INFORMATICS DEPARTMENT

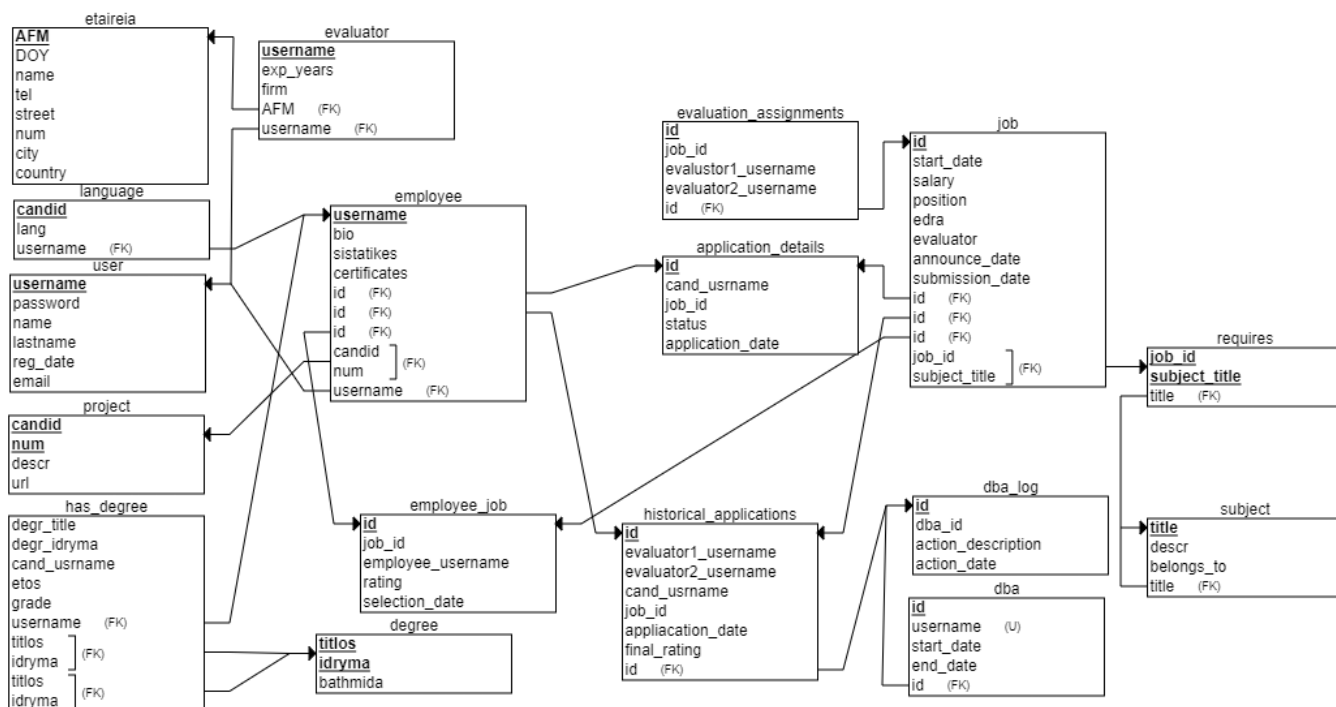
Εργαστήριο Βάσεις
Δεδομένων:Project
2023-2024

ΟΝΟ/ΜΟ: ΓΕΟΔΩΡΟΣ ΚΑΤΣΑΝΤΑΣ
ΑΜ: 1097459
Έτος Φοίτησης: 3^ο

ΟΝΟ/ΜΟ: ΑΓΓΕΛΙΚΗ ΔΟΥΒΡΗ
ΑΜ: 1097441
Έτος Φοίτησης: 3^ο

Περιεχόμενα

Κεφάλαιο 1σελ 3	
Ερώτημα 3.1.1.σελ 3	
Ερώτημα 3.1.2 Νέες απαιτήσεις.....σελ 10	
Ερώτημα 3.1.2.1.....σελ 11	
Ερώτημα 3.1.2.2.....σελ 12	
Ερώτημα 3.1.2.3.....σελ 16	
Ερώτημα 3.1.2.4.....σελ 17	
Ερώτημα 3.1.3 Δημιουργία Stored Procedure.....σελ 18	
Ερώτημα 3.1.3.1.....σελ 18	
Ερώτημα 3.1.3.2.....σελ 19	
Ερώτημα 3.1.3.3.....σελ 20	
Ερώτημα 3.1.3.4.....σελ 21	
Ερώτημα 3.1.4.1.....σελ 22	
Ερώτημα 3.1.4.2.....σελ 23	
Ερώτημα 3.1.4.3.....σελ 24	
Κεφάλαιο 2σελ 25	
Ερώτημα 3.1.3.1.....σελ 25	
Ερώτημα 3.1.3.2.....σελ 26	
Ερώτημα 3.1.3.3.....σελ 29	
Ερώτημα 3.1.3.4.....σελ 32	
Κεφάλαιο 3σελ 36	
Ερώτημα 3.1.4.1σελ 36	
Ερώτημα 3.1.4.2.....σελ 40	
Ερώτημα 3.1.4.3.....σελ 42	
Δημιουργία Γραφικών Διεπαφών Χρήστη 3.2.1.σελ 44	
Περιγραφή λειτουργικότητας.....σελ 44	
Ένωση με την βάση mysql.....σελ 44	
Γενικός Σχεδιασμός.....σελ 44	
Log in pageσελ 45	
Λεπτομέρειες στα JFrames.....σελ 46	



Κεφάλαιο 1

Το παραπάνω αποτελεί το σχεσιακό διάγραμμα της Βάσης Δεδομένων μας.

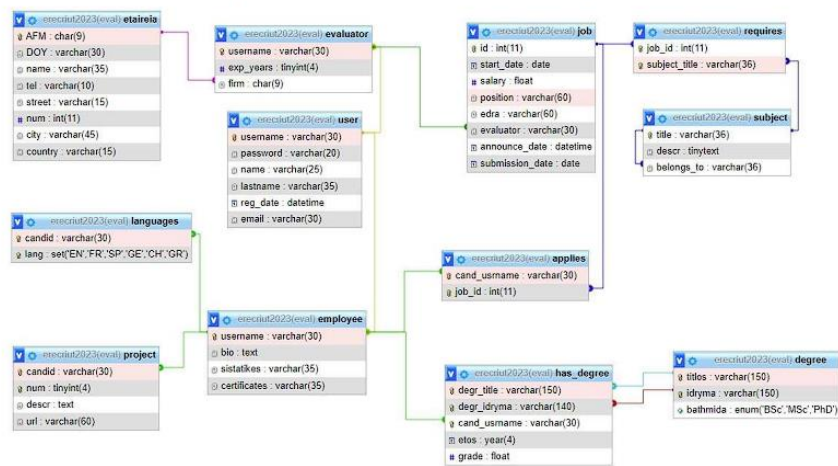
Ακολουθεί η αναλυτική επεξήγηση της υλοποίησης κάθε ερωτήματος, καθώς και ο κώδικας που αντιστοιχεί. Για ενδεχόμενο έλεγχο ο κώδικας θα υπάρχει επίσης ενιαίος στο τέλος του Κεφαλαίου 1.

*Διευκρίνιση: Η επεξήγηση της υλοποίησης και λειτουργίας των Stored **PROCEDURES**, Triggers θα βρίσκεται επίσης στο Κεφάλαιο 1, ενώ ο αντίστοιχος κώδικας και τα παραδείγματα τους στο Κεφάλαιο 2 και 3 αντίστοιχα.

Ερώτημα 3.1.1

Στην προπαρασκευαστική φάση κληθήκαμε να δημιουργήσουμε την ακόλουθη Βάση Δεδομένων και να εισάγουμε σε αυτή ένα πλήθος εγγγραφών σε κάθε πίνακα.

Αρχικά, λοιπόν, κατασκευάσαμε τη βάση μας και δημιουργήσαμε τους πίνακες applies, degree, employee, etaireia, evaluator, has_degree, job, languages, project, requires, subject, user όπως φαίνεται στο κομμάτι του κώδικα που ακολουθεί:



ΠΡΟΠΑΡΑΣΚΕΥΑΣΤΙΚΟ ΣΤΑΔΙΟ-ΕΡΩΤΗΜΑ 3.1.1 --

DROP DATABASE if exists erecruit2023;

CREATE DATABASE erecruit2023;

USE erecruit2023;

```
CREATE TABLE IF NOT EXISTS etaireia (
    AFM CHAR(9) DEFAULT 'unknown' NOT NULL,
    DOY VARCHAR(30) DEFAULT 'unknown' NOT NULL,
    name VARCHAR(35) DEFAULT 'unknown' NOT NULL,
    tel VARCHAR(10) DEFAULT '0' NOT NULL,
    street VARCHAR(15) DEFAULT 'unknown' NOT NULL,
    num INT(11) DEFAULT '0' NOT NULL,
    city VARCHAR(45) DEFAULT 'unknown' NOT NULL,
    country VARCHAR(15) DEFAULT 'unknown' NOT NULL,
    PRIMARY KEY (AFM)
);
```

```
CREATE TABLE IF NOT EXISTS user(
    username VARCHAR(30) DEFAULT 'unknown' NOT NULL,
    password VARCHAR(20) DEFAULT 'unknown' NOT NULL,
    name VARCHAR(25) DEFAULT 'unknown' NOT NULL,
    lastname VARCHAR(35) DEFAULT 'unknown' NOT NULL,
    reg_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    email VARCHAR(30) DEFAULT 'unknown' NOT NULL,
    PRIMARY KEY(username)
);
```

```
CREATE TABLE IF NOT EXISTS evaluator(
    username VARCHAR(30) DEFAULT 'unknown' NOT NULL,
    exp_years TINYINT(4) DEFAULT '0' NOT NULL,
```

```

        firm CHAR(9) DEFAULT 'unknown' NOT NULL,
        PRIMARY KEY(username),
        CONSTRAINT eval_etaireia FOREIGN KEY (firm) REFERENCES etaireia(AFM)
        ON DELETE CASCADE ON UPDATE CASCADE,
        CONSTRAINT eval_username FOREIGN KEY (username) REFERENCES user(username)
        ON DELETE CASCADE ON UPDATE CASCADE
    );

```

```

CREATE TABLE IF NOT EXISTS subject(
    title VARCHAR(36) DEFAULT 'unknown' NOT NULL,
    descr TINYTEXT NOT NULL,
    belongs_to VARCHAR(36) DEFAULT NULL,
    PRIMARY KEY(title),
    CONSTRAINT subjbelongs FOREIGN KEY (belongs_to) REFERENCES subject(title)
    ON DELETE CASCADE ON UPDATE CASCADE
);

```

```

CREATE TABLE IF NOT EXISTS job (
    id INT(11) AUTO_INCREMENT NOT NULL,
    start_date DATE NOT NULL,
    salary FLOAT DEFAULT '0' NOT NULL,
    position VARCHAR(60) DEFAULT 'unknown' NOT NULL,
    edra VARCHAR(60) DEFAULT 'unknown' NOT NULL,
    evaluator VARCHAR(30) DEFAULT 'unknown' NOT NULL,
    announce_DATE DATETIME DEFAULT current_timestamp(),
    submission_DATE DATE NOT NULL ,
    PRIMARY KEY(id),
    CONSTRAINT eval_job FOREIGN KEY (evaluator) REFERENCES evaluator(username)
    ON DELETE CASCADE ON UPDATE CASCADE
);

```

```

CREATE TABLE IF NOT EXISTS requires (
    job_id INT(11) DEFAULT '0' NOT NULL,
    subject_title VARCHAR(36) DEFAULT 'unknown' NOT NULL,
    PRIMARY KEY (job_id, subject_title),
    CONSTRAINT requires_job FOREIGN KEY (job_id) REFERENCES job(id)

```

ON DELETE CASCADE ON UPDATE CASCADE

);

```
CREATE TABLE IF NOT EXISTS employee(  
    username VARCHAR(30) DEFAULT 'unknown' NOT NULL,  
    bio TEXT NOT NULL,  
    sistatikes VARCHAR(35) DEFAULT 'unknown' NOT NULL,  
    certificates VARCHAR(35) DEFAULT 'unknown' NOT NULL,  
    PRIMARY KEY(username) ,  
    CONSTRAINT empl_username FOREIGN KEY (username) REFERENCES user(username)  
    ON DELETE CASCADE ON UPDATE CASCADE
```

);

```
CREATE TABLE IF NOT EXISTS languages(  
    candid VARCHAR(30) DEFAULT 'unknown' NOT NULL,  
    lang SET ('EN', 'FR', 'SP', 'GE', 'CH', 'GR'),  
    PRIMARY KEY(candid,lang),  
    CONSTRAINT candid_languages FOREIGN KEY (candid) REFERENCES employee(username)
```

);

```
CREATE TABLE IF NOT EXISTS project(  
    candid VARCHAR(30) DEFAULT 'unknown' NOT NULL,  
    num TINYINT(4) DEFAULT '0' NOT NULL,  
    descr TEXT NOT NULL,  
    url VARCHAR(60) DEFAULT 'unknown' NOT NULL,  
    PRIMARY KEY(candid,num),  
    CONSTRAINT candid_project FOREIGN KEY (candid) REFERENCES employee(username)
```

);

```
CREATE TABLE IF NOT EXISTS degree (  
    titlos VARCHAR(150) DEFAULT 'unknown' NOT NULL,  
    idryma VARCHAR(140) DEFAULT 'unknown' NOT NULL,  
    bathmida ENUM('BSc','MSc','PhD'),  
    PRIMARY KEY(titlos, idryma)
```

);

```
CREATE TABLE IF NOT EXISTS has_degree(  
    degr_title VARCHAR(150) DEFAULT 'unknown' NOT NULL,  
    degr_idryma VARCHAR(140) DEFAULT 'unknown' NOT NULL,  
    cand_username VARCHAR(30) DEFAULT 'unknown' NOT NULL,  
    etos YEAR(4) DEFAULT '0' NOT NULL,  
    grade FLOAT DEFAULT '0' NOT NULL,  
    PRIMARY KEY(degr_title, degr_idryma, cand_username),  
    CONSTRAINT has_degree_username FOREIGN KEY (degr_title, degr_idryma) REFERENCES degree(titlos,  
idryma),  
    CONSTRAINT has_degree1_username FOREIGN KEY (cand_username) REFERENCES  
employee(username)  
);
```

```
CREATE TABLE IF NOT EXISTS applies (  
    cand_username VARCHAR(30) DEFAULT 'unknown' NOT NULL,  
    job_id INT(11) DEFAULT '0' NOT NULL,  
    PRIMARY KEY (cand_username, job_id),  
    CONSTRAINT applies_username FOREIGN KEY (cand_username) REFERENCES employee(username)  
ON DELETE CASCADE ON UPDATE CASCADE,  
    CONSTRAINT applies_job FOREIGN KEY (job_id) REFERENCES job(id)  
ON DELETE CASCADE ON UPDATE CASCADE  
);
```

Έπειτα, έγινε χρήση των κατάλληλων εντολών **INSERT** για να γεμίσουμε τους πίνακες μας όπως φαίνεται παρακάτω:

```
INSERT INTO etaireia (AFM, DOY, name, tel, street, num, city, country) VALUES  
('265032148','Patrwn', 'Nikolaou', '2610486522' , 'Mezonos', '59', 'Patras', 'Greece'),  
('623946215', 'Patrwn', 'Papadopoulos kai sia', '2103625577', 'Ermou', '150', 'Athens', 'Greece'),
```

```
('203492781', 'Patrwn', 'Balafouths AE', '2610885423', 'Agiou Nikolaou', '23', 'Patras', 'Greece');
```

```
INSERT INTO employee (username, bio, sistatikes, certificates) VALUES
```

```
('Xristopoulos', 'XristopoulosN.docx', 'sistatikhXRIST.docx', 'pistXristopoulos.docx'),  
( 'Lious', 'Lious.docx', 'sistatikhLious.docx', 'LiousCertificates.docx'),  
( 'Papanikolaou', 'Papanikolaou.docx', 'sistatikhPapanikolaou.docx', 'certPapanikolaou.docx'),  
( 'Nikolakopoulou', 'Nikolakopoulou.pdf', 'sistatikhMarianna.pdf', 'DEFAULT'),  
( 'Ioannidh', 'Ioannidh.pdf', 'DEFAULT', 'IoannidhCertificates.pdf'),  
( 'Papapaulou', 'PapapaulouBio.pdf', 'PapapaulouSist.pdf', 'DEFAULT');
```

```
INSERT INTO evaluator (username, exp_years, firm) VALUES
```

```
('Douvris', '15', '265032148'),  
( 'Georgiou', '20', '623946215'),  
( 'Androutsos', '14', '203492781'),  
( 'Dhnhtriou', '6', '265032148'),  
( 'Andrikopoulos', '2', '203492781'),  
( 'Katsaros', '17', '623946215');
```

```
INSERT INTO degree (titlos, idryma, bathmida) VALUES
```

```
('Computer Engineering and Informatics', 'Panepisthmio Patrwn', 'BSc'),  
( 'Electrical Engineering', 'Panepisthmio Patrwn', 'MSc'),  
( 'Mathematics', 'Panepisthmio Patrwn', 'BSc'),  
( 'Logistics', 'Panepisthmio Patrwn', 'BSc'),  
( 'Architecture', 'Panepisthmio Patrwn', 'PhD'),  
( 'Economics', 'Panepisthmio Patrwn', 'MSc');
```

```
INSERT INTO job (start_date, salary, position, edra, evaluator, announce_date, submission_date) VALUES
```

```
('2022-01-01', '67000', 'Ypeuthinos C', 'Patra', 'Androutsos', '2022-01-01', '2024-12-31' ),  
( '2022-02-01', '40000', 'Ypeuthinos G', 'Patra', 'Georgiou', '2022-02-01', '2024-12-31' ),  
( '2022-03-01', '102000', 'Ypeuthinos A', 'Patra', 'Douvris', '2022-03-01', '2024-12-31' ),  
( '2022-04-01', '54000', 'Ypeuthinos F', 'Patra', 'Androutsos', '2022-04-01', '2024-12-31'),  
( '2022-05-01', '65000', 'Ypeuthinos D', 'Patra', 'Dhnhtriou', '2022-05-01', '2024-12-31' ),  
( '2022-06-01', '58000', 'Ypeuthinos E', 'Patra', 'Andrikopoulos', '2022-06-01', '2024-12-31' ),  
( '2022-07-01', '92000', 'Ypeuthinos B', 'Patra', 'Katsaros', '2022-07-01', '2024-12-31' ),  
( '2022-08-01', '58000', 'Ypeuthinos E', 'Patra', 'Katsaros', '2022-08-01', '2024-12-31' );
```

```
INSERT INTO has_degree (degr_title, degr_idryma, cand_usrname, etos, grade) VALUES
```



```
( 'Computer Engineering and Informatics', 'Panepisthmio Patrwn', 'Xristopoulos', '2023', '8'),
( 'Economics', 'Panepisthmio Patrwn', 'Liious', '2022', '9'),
( 'Architecture', 'Panepisthmio Patrwn', 'Nikolakopoulou', '2023', '8.6'),
( 'Electrical Engineering', 'Panepisthmio Patrwn', 'Papapaulou', '2023', '6');
```

INSERT INTO languages (candid, lang) VALUES

```
( 'Xristopoulos', 'EN'),
( 'Liious', 'GR'),
( 'Nikolakopoulou', 'SP'),
( 'Papapaulou', 'CH');
```

INSERT INTO project (candid, num, descr, url) VALUES

```
( 'Xristopoulos', '1', 'xristopoulos.pdf', 'https://XRproject.com' ),
( 'Liious', '2', 'liious1.pdf', 'https://LI1project.com' ),
( 'Liious', '3', 'liious2.pdf', 'https://LI2project.com' ),
( 'Papanikolaou', '4', 'papanikolaou.pdf', 'https://PAPAPproject.com' ),
( 'Nikolakopoulou', '5', 'nikolakopoulou.pdf', 'https://NIKOLproject.com' ),
( 'Ioannidh', '6', 'ioannidh.pdf', 'https://IOANproject.com' ),
( 'Papapaulou', '7', 'papapaulou.pdf', 'https://PAPAPAPproject.com' );
```

INSERT INTO user (username, password, name, lastname, reg_date, email) VALUES

```
( 'Douvris', 'Jdjoswkds', 'Fotis', 'Douvris', '2009-12-14', 'douvrisf@yahoo.com'),
( 'Georgiou', '15121996', 'Aggelikh', 'Georgiou', '2004-06-06', 'aggelikhgeorgiou@gmail.com'),
( 'Androutsos', 'Kddkdekl', 'Thodorhs', 'Androutsos', '2010-01-09', 'thodandrou@gmail.com'),
( 'Dhbmhtriu', 'Panagiota29', 'Panagiota', 'Dhbmhtriu', '2018-05-25',
'panagiotadhbmhtriu@gmail.com'),
( 'Andrikopoulos', 'Lakksood', 'Dhbmhtrhs', 'Andrikopoulos', '2022-02-22',
'megalosdhbmhtrhs@yahoo.com'),
( 'Katsaros', '20202020', 'Alejandros', 'Katsaros', '2007-11-07', 'alejandroskats@gmail.com'),
( 'Xristopoulos', 'Xmlmjkd', 'Nikolas', 'Xristopoulos', '2022-01-01', 'nikolasxrist@gmail.com'),
( 'Liious', 'Fr@nk1i0us', 'Frank', 'Liious', '2023-02-01', 'liiousfrank@gmail.com'),
( 'Papanikolaou', 'Nt1n@26', 'Konstantina', 'Papanikolaou', '2022-03-01',
'ntinapapa@yahoo@gmail.com'),
( 'Nikolakopoulou', 'Anna1985', 'Marianna', 'Nikolakopoulou', '2022-04-01',
'Marianna1985@yahoo.com'),
( 'Ioannidh', '56426348', 'Hlianna', 'Ioannidh', '2022-05-01', 'ioannidhhl@yahoo.com'),
( 'Papapaulou', 'Mariamaria', 'Maria', 'Papapaulou', '2022-06-01', 'Mariapapa@gmail.com');
```

INSERT INTO subject (title, descr, belongs_to) VALUES

('Web Developer', 'This job requires experience in designing, developing, and maintaining web applications.', null),

('Product Manager', 'This job requires experience in defining, prioritizing, and launching new products.', null),

('Marketing Specialist', 'This job requires experience in creating and executing marketing campaigns to promote products and services.', null),

('Customer Support Specialist', 'This job requires experience in providing technical and customer support to resolve issues and answer questions.', null),

('Accountant', 'This job requires experience in managing financial records, preparing reports, and ensuring financial compliance.', null),

('Human Resources Manager', 'This job requires experience in recruiting, onboarding, and managing employees.', null);

INSERT INTO requires (job_id, subject_title) VALUES

('1','Web Development'),

('2','Marketing Specialist'),

('3','Human Resources Manager'),

('4','Web Development'),

('5','Accountant'),

('6','Customer Support Specialist'),

('7','Product Manager'),

('8','Web Development');

INSERT INTO applies (cand_username, job_id) VALUES

('Xristopoulos', '3'),

('Lious', '1'),

('Papanikolaou', '3'),

('Nikolakopoulou', '5'),

('Ioannidh', '3'),

('Papapaulou', '8');

Ερώτημα 3.1.2 Νέες απαιτήσεις

Καλούμαστε σε αυτό το σημείο να τροποποιήσουμε είτε να αναπτύξουμε περαιτέρω τη Βάση μας ώστε να καλύψει τις απαιτήσεις που μας ζητήθηκαν. Μετά την υλοποίηση κάθε πίνακα θα πραγματοποιείται ένα **INSERT**, όπου είναι απαραίτητο, για να έχουμε κατάλληλο πλήθος εγγραφών. Τέλος, θα τεκμηριώσουμε γιατί δράσαμε όπως δράσαμε.

Ερώτημα 3.1.2.1

Αρχίζοντας, λοιπόν, σκοπός μας είναι να κατασκευάσουμε έναν μηχανισμό ο οποίος θα διαχειρίζεται τις αιτήσεις. Πρώτο μας μέλημα ήταν η τροποποίηση του πίνακα **applies**, στον οποίο προστέθηκαν οι στήλες **id**, **status**, **application_date**, ενώ αλλάξαμε και το όνομα του σε **application_details**. Ακολουθεί ο κώδικας που χρειάστηκε για αυτή την ενέργεια:

```
-- DROP FOREIGN KEY CONSTRAINTs
```

```
ALTER TABLE applies
```

```
DROP FOREIGN KEY applies_username,
```

```
DROP FOREIGN KEY applies_job;
```

```
-- DROP the existing PRIMARY KEY
```

```
ALTER TABLE applies
```

```
DROP PRIMARY KEY;
```

```
-- ADD a new column 'id' with AUTO_INCREMENT and make it the new PRIMARY KEY
```

```
ALTER TABLE applies
```

```
ADD COLUMN id INT(11) AUTO_INCREMENT PRIMARY KEY FIRST;
```

```
-- ADD new columns 'status' and 'application_date'
```

```
ALTER TABLE applies
```

```
ADD COLUMN status ENUM('active', 'completed', 'cancelled') DEFAULT 'active' NOT NULL AFTER job_id,
```

```
ADD COLUMN application_date DATETIME DEFAULT CURRENT_TIMESTAMP AFTER status;
```

```
-- ADD new FOREIGN KEY CONSTRAINTs on the 'cand_username' and 'job_id' columns
```

```
ALTER TABLE applies
```

```
-- Rename the TABLE to 'application_details'  
RENAME TABLE applies TO application_details;
```

```
/* The new applies TABLE  
CREATE TABLE IF NOT EXISTS application_details (  
    id INT(11) AUTO_INCREMENT NOT NULL,  
    cand_username VARCHAR(30) DEFAULT 'unknown' NOT NULL,  
    job_id INT(11) DEFAULT '0' NOT NULL,  
    status ENUM('active', 'completed', 'cancelled') DEFAULT 'active' NOT NULL,  
    application DATE DATETIME DEFAULT CURRENT_TIMESTAMP,  
    PRIMARY KEY (id),  
    CONSTRAINT app_details_username FOREIGN KEY (cand_username) REFERENCES employee(username)  
    ON DELETE CASCADE ON UPDATE CASCADE,  
    CONSTRAINT app_details_job FOREIGN KEY (job_id) REFERENCES job(id)  
    ON DELETE CASCADE ON UPDATE CASCADE  
);  
*/
```

Εφόσον έχει δημιουργηθεί αυτός ο ‘καινούριος’ πίνακας μπορούμε τώρα να τον χρησιμοποιήσουμε. Μας ζητήθηκε να φτιάξουμε έναν μηχανισμό για την εισαγωγή και τη διαχείριση αιτήσεων προαγωγής. Το πρώτο απαιτούμενο καλύφθηκε από τον νέο πίνακα με τη χρήση του **ENUM**. Τα άλλα 3 απαιτούμενα χωρίζονται σε 2 κατηγορίες:

- 1) Έλεγχος κατά την εισαγωγή μιας αίτησης
- 2) Τροποποίηση ήδη υπάρχουσας αίτησης

Γι’ αυτό το λόγο θα δημιουργήσουμε 2 διαφορετικά triggers, το `before_insert_application` και το `before_update_application`. Το 1^ο trigger ελέγχει αν μία αίτηση γίνεται εντός 15 ημερών από την ημερομηνία έναρξης της θέσης, αν ναι τότε εμφανίζει αντίστοιχο μήνυμα και δεν το επιτρέπει. Επίσης, αν εντοπίσει ότι ο υποψήφιος έχει ήδη τρεις αιτήσεις δεν επιτρέπει νέα εισαγωγή.

Το 2^ο trigger ελέγχει τι συμβαίνει αν προσπαθήσουμε να ακυρώσουμε μια αίτηση εντός 10 ημερών πριν την έναρξη της θέσης. Επίσης, αν θέλουμε να μετατρέψουμε μία αίτηση από ‘cancelled’ σε ‘active’ γίνεται έλεγχος εάν υπάρχουν 3 αιτήσεις για αυτό το άτομο. Και στις 2 περιπτώσεις εμφανίζεται αντίστοιχο error. Ο κώδικας των 2 triggers βρίσκεται στο Κεφάλαιο 3 μαζί με αντίστοιχα παραδείγματα που δείχνουν τη λειτουργικότητα του (Ερωτήματα 3.1.4.2 & 3.1.4.3).

Ερώτημα 3.1.2.2

Πρόκειται για ένα από τα πιο σύνθετα ερωτήματα του Project με πολλές απαιτήσεις.

Για αρχή δημιουργήσαμε 2 νέους πίνακες, τον evaluation_assignments ο οποίος μας δείχνει ποιοι 2 evaluators έχουν αναλάβει την αξιολόγηση των αιτήσεων για μία θέση εργασίας(του κάναμε INSERT τους evaluators γιατί θα χρειαστεί παρακάτω να τους γνωρίζουμε) και τον historical_applications στον οποίο αποθηκεύονται όλες οι αιτήσεις που έχουν επεξεργαστεί.

```
CREATE TABLE IF NOT EXISTS evaluation_assignments (  
  id INT(11) AUTO_INCREMENT NOT NULL,  
  job_id INT(11) NOT NULL,  
  evaluator1_username VARCHAR(30) NOT NULL,  
  evaluator2_username VARCHAR(30) NOT NULL,  
  PRIMARY KEY (id),  
  CONSTRAINT eval_assign_job FOREIGN KEY (job_id) REFERENCES job(id) ON DELETE CASCADE ON  
  UPDATE CASCADE  
);
```

-- Inserting evaluators for a specific job ID

```
INSERT INTO evaluation_assignments (job_id, evaluator1_username, evaluator2_username)  
VALUES  
  (1, 'Andrikopoulos', 'Androutsos'),  
  (2, 'Dhnhtriou', 'Douvris'),  
  (3, 'Georgiou', 'Katsaros'),  
  (4, 'Douvris', 'Georgiou'),  
  (5, 'Katsaros', 'Androutsos'),  
  (6, 'Dhnhtriou', 'Andrikopoulos');
```

```
CREATE TABLE IF NOT EXISTS historical_applications (  
  id INT(11) AUTO_INCREMENT NOT NULL,  
  evaluator1_username VARCHAR(30) DEFAULT 'unknown' NOT NULL,  
  evaluator2_username VARCHAR(30) DEFAULT 'unknown' NOT NULL,  
  cand_username VARCHAR(30) DEFAULT 'unknown' NOT NULL,  
  job_id INT(11) DEFAULT '0' NOT NULL,  
  status ENUM('active', 'completed', 'canceled') DEFAULT 'completed' NOT NULL,  
  application_date DATETIME DEFAULT CURRENT_TIMESTAMP,  
  final_rating INT NOT NULL DEFAULT 0,
```

```

PRIMARY KEY (id),
CONSTRAINT hist_app_username FOREIGN KEY (cand_username) REFERENCES employee(username)
ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT hist_app_job FOREIGN KEY (job_id) REFERENCES job(id)
ON DELETE CASCADE ON UPDATE CASCADE
);

```

Επόμενη ενέργεια είναι να φτιάξουμε ένα stored **PROCEDURE** το οποίο ελέγχει την κατάσταση μίας αίτησης ούτως ώστε να την καλέσουμε στο stored **PROCEDURE** που θα υπολογίζει την βαθμολογία της αίτησης.

```

DELIMITER //

```

```

CREATE PROCEDURE check_application_status(IN candidate_username VARCHAR(30), IN job_id INT,
OUT app_status VARCHAR(20))

```

```

BEGIN

```

```

-- Get the status of the application

```

```

SELECT COALESCE(status, 'not_found')

```

```

INTO app_status

```

```

FROM application_details

```

```

WHERE cand_username = candidate_username AND job_id = job_id

```

```

LIMIT 1;

```

```

END //

```

```

DELIMITER ;

```

Ο μηχανισμός απαιτεί να αγνοούνται οι ακυρωμένες αιτήσεις, καθεμία να αξιολογείται από δύο βαθμολογητές(ελέγχεται μέσω του πίνακα evaluation_assignments αν κάποιος έχει αναλάβει συγκεκριμένο id) με βαθμούς από 1 έως 20, όταν δεν υπάρχουν και οι 2 βαθμοί γίνονται ανάλογες ενέργειες που περιγράφονται στην stored για να συμπληρωθούν και τέλος οι αιτήσεις που έχουν επεξεργαστεί αφαιρούνται από τον application_details και μεταφέρονται στον historical_applications. Όλα αυτά υλοποιούνται από την evaluate_promotion η οποία είναι μια εκτενής και λεπτόμερης διαδικασία:

```

DELIMITER //

```

```

CREATE PROCEDURE evaluate_promotion(

```

```

IN candidate_username VARCHAR(30),
IN job_id INT
)
BEGIN
    DECLARE evaluator1_username VARCHAR(30);
    DECLARE evaluator2_username VARCHAR(30);
    DECLARE rating1 INT;
    DECLARE rating2 INT;
    DECLARE final_rating INT;
    DECLARE app_status VARCHAR(20);
    DECLARE job_exists INT;

    -- Check if the job_id exists IN the applications
    SELECT COUNT(*)
    INTO job_exists
    FROM application_details
    WHERE job_id = job_id;

    -- If the job_id does not exist, prINT an error message and exit the PROCEDURE
    IF job_exists = 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Job does not exist in applications. Skipping evaluation.';
    END IF;

    -- Check if evaluators are assigned for the given job_id
    SELECT ea.evaluator1_username, ea.evaluator2_username
    INTO evaluator1_username, evaluator2_username
    FROM evaluation_assignments ea
    WHERE ea.job_id = job_id
    LIMIT 1;

    -- If evaluators are not assigned, prINT an error message and exit the PROCEDURE
    IF evaluator1_username IS NULL OR evaluator2_username IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Evaluators are not assigned for the given job_id. Skipping evaluation.';
    END IF;

```

```

-- Call the PROCEDURE to check the application status
CALL check_application_status(candidate_username, job_id, app_status);

-- Check if the application is cancelled or completed
IF TRIM(app_status) = 'cancelled' THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Application is cancelled. Skipping evaluation.';
ELSEIF TRIM(app_status) = 'completed' THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Application is already completed. Skipping evaluation.';
ELSE
    -- Generate random ratings between 1 and 20 for both evaluators, with a possibility of NULL
    SET rating1 = (CASE WHEN RAND() < 0.8 THEN FLOOR(1 + RAND() * 20) ELSE NULL END);
    SET rating2 = (CASE WHEN RAND() < 0.8 THEN FLOOR(1 + RAND() * 20) ELSE NULL END);

    -- Calculate the final rating based on qualifications
    SELECT (SUM(IF(dg.bathmida = 'BSc', 1, IF(dg.bathmida = 'MSc', 2, IF(dg.bathmida = 'PhD', 3, 0)))) +
        (SELECT COUNT(*) FROM languages l WHERE l.candid = candidate_username) +
        (SELECT COUNT(*) FROM project p WHERE p.candid = candidate_username))
    INTO final_rating
    FROM has_degree hd
    JOIN degree dg ON hd.degr_title = dg.titlos AND hd.degr_idryma = dg.idryma
    WHERE hd.cand_usrname = candidate_username;

    -- If one rating is missing, fill in based on qualifications
    IF rating1 IS NULL AND rating2 IS NOT NULL THEN
        SET rating1 = ((SUM(IF(dg.bathmida = 'BSc', 1, IF(dg.bathmida = 'MSc', 2, IF(dg.bathmida = 'PhD', 3,
0)))) +
            (SELECT COUNT(*) FROM languages l WHERE l.candid = candidate_username) +
            (SELECT COUNT(*) FROM project p WHERE p.candid = candidate_username)));
    END IF;
    IF rating2 IS NULL AND rating1 IS NOT NULL THEN
        SET rating2 = ((SUM(IF(dg.bathmida = 'BSc', 1, IF(dg.bathmida = 'MSc', 2, IF(dg.bathmida = 'PhD', 3,
0)))) +
            (SELECT COUNT(*) FROM languages l WHERE l.candid = candidate_username) +
            (SELECT COUNT(*) FROM project p WHERE p.candid = candidate_username)));
    END IF;

```



```

IF rating1 IS NULL AND rating2 IS NULL THEN
    -- Calculate the DEFAULT rating based on qualifications
    SET final_rating = ((SUM(IF(dg.bathmida = 'BSc', 1, IF(dg.bathmida = 'MSc', 2, IF(dg.bathmida =
'PhD', 3, 0)))) +
        (SELECT COUNT(*) FROM languages l WHERE l.candid = candidate_username) +
        (SELECT COUNT(*) FROM project p WHERE p.candid = candidate_username));
ELSE
    -- Use the provided ratings if available
    SET final_rating = ((COALESCE(rating1, 0) + COALESCE(rating2, 0)) / 2);
END IF;

-- Set session variables for the ratings and final rating
SET @rating1 = rating1;
SET @rating2 = rating2;
SET @final_rating = final_rating;

-- Print the ratings and final rating, including the candidate's username
SELECT candidate_username AS 'Candidate Username', @rating1 AS 'Evaluator 1 Rating', @rating2
AS 'Evaluator 2 Rating', @final_rating AS 'Final Rating';
END IF;
END //

```

DELIMITER;

*Διευκρίνιση: Οι βαθμοί των evaluators εισάγονται τυχαία με τη χρήση της RAND, με πιθανότητα 20% να είναι NULL για να καλυφθούν όλες οι περιπτώσεις

Σαφώς δεν ξεχάσαμε την απαίτηση για την εξαγωγή αποτελέσματος μίας θέσης, αυτή υλοποιείται με μία νέα stored **PROCEDURE** την fill_job_position η οποία χρησιμοποιεί την evaluate_promotion. Περισσότερα για αυτή θα δούμε στο Ερώτημα 3.1.3.3

Ερώτημα 3.1.2.3

Ο πίνακας historical_applications έχει υλοποιηθεί ήδη οπότε το μόνο που μένει είναι να καταχωρήσουμε τις 60.000 εγγραφές, εφόσον δεν μας ενδιαφέρουν οι περιορισμοί τότε αποφασίσαμε να φτιάξουμε ένα **PROCEDURE** που θα εισάγει σε αυτόν τον πίνακα εγγραφές σε ένα loop μέχρι να ξεπεράσει τις 60.000. Η εισαγωγή γίνεται σε batches για να γίνονται πετυχημένα όλες χωρίς να κρασάρει. Ακολουθεί ο κώδικας της stored:

```

DELIMITER //
CREATE PROCEDURE generate_historical_applications()
BEGIN
    DECLARE i INT DEFAULT 0;
    DECLARE batch_size INT DEFAULT 500;

    WHILE i <= 60000 DO
        START TRANSACTION;

        -- INSERT INTO historical_applications with both evaluators
        INSERT INTO historical_applications (evaluator1_username, evaluator2_username, cand_username,
        job_id, status, final_rating)
        SELECT
            (SELECT evaluator1_username FROM evaluation_assignments ORDER BY RAND() LIMIT 1) AS
            evaluator1_username,
            (SELECT evaluator2_username FROM evaluation_assignments WHERE evaluator2_username <>
            evaluator1_username ORDER BY RAND() LIMIT 1) AS evaluator2_username,
            (SELECT username FROM employee ORDER BY RAND() LIMIT 1) AS cand_username,
            (SELECT id FROM job ORDER BY RAND() LIMIT 1) AS job_id,
            'completed' AS status,
            FLOOR(1 + RAND() * 20) AS final_rating
        FROM information_schema.tables ORDER BY RAND() LIMIT batch_size;

        COMMIT;

        SET i = i + batch_size;
    END WHILE;
END //
DELIMITER ;

```

Ερώτημα 3.1.2.4

Για αυτή την απαίτηση θα φτιάξουμε νέο πίνακα με όνομα dba (aka Database Admins).
Με βάση τα ζητούμενα ο πίνακας θα φτιαχτεί έτσι:

```

CREATE TABLE IF NOT EXISTS dba (
    id INT(11) AUTO_INCREMENT NOT NULL,
    username VARCHAR(30) NOT NULL,

```

```
start_date DATE NOT NULL,  
end_date DATE,  
PRIMARY KEY (id),  
UNIQUE KEY (username)  
);  
  
INSERT INTO dba (username, start_date) VALUES ('theokatsa', '2024-01-10');  
INSERT INTO dba (username, start_date) VALUES ('aggidouvri', '2024-01-10');  
*Έγινε εισαγωγή 2 admins καθώς θα χρειαστούν αργότερα
```

Ερώτημα 3.1.3 Δημιουργία Stored Procedure

Όπως έχει αναφερθεί και νωρίτερα, σε αυτό το κομμάτι θα αναλύσουμε την δημιουργία των **PROCEDURES**, ενώ ο αντίστοιχος κώδικας για την καθεμία θα βρίσκεται στο Κεφάλαιο 2 μαζί με παραδείγματα εκτέλεσης της.

Ερώτημα 3.1.3.1

Είναι προφανές πως σε αυτό το ερώτημα θα χρειαστούμε τους πίνακες evaluator, employee, evaluation_assignments και historical_applications. Πρέπει το **PROCEDURE** να ελέγχει αν η θέση εργασίας αντιστοιχεί στον αξιολογητή, αν υπάρχει ήδη βαθμός και αν όχι να υπολογίζεται με χρήση της evaluate_promotion από το Ερώτημα 3.1.2.2. Έχουν οριστεί αντίστοιχα error messages ανάλογα την περίπτωση ούτως ώστε να τυπωθεί και το αντίστοιχο αποτέλεσμα.

Αναλυτικότερα, η CheckEvaluation λειτουργεί ως εξής:

- 1) Ορισμός παραμέτρων (p_evaluator_username, p_employee_username, p_job_id , p_result)
- 2) Δήλωση μεταβλητής message_text που χρησιμοποιείται για την αποθήκευση μηνυμάτων.
- 3) Έλεγχος ύπαρξης αξιολογητή για τη θέση εργασίας μέσω ενός ελέγχου **SELECT COUNT(*)** για να δει αν ο αξιολογητής (p_evaluator_username) υπάρχει για τη συγκεκριμένη θέση εργασίας (p_job_id) στον πίνακα evaluation_assignments και το αποτέλεσμα του αποθηκεύεται στην παράμετρο εξόδου p_result.
- 4) Έλεγχος ύπαρξης υπαλλήλου (p_employee_username) στον πίνακα employee. Εάν ο υπάλληλος δεν υπάρχει, εμφανίζεται ένα SQLSTATE '45000' error με το αντίστοιχο μήνυμα.
- 5) Έλεγχος ύπαρξης αξιολόγησης: Αν ο υπάλληλος υπάρχει, τότε ελέγχεται εάν υπάρχει ήδη αξιολόγηση για τον συγκεκριμένο αξιολογητή, υπάλληλο και θέση

εργασίας στον πίνακα `historical_applications`. Εάν υπάρχει, εμφανίζεται ένα `SQLSTATE '45000' error` με το αντίστοιχο μήνυμα.

- 6) Υπολογισμός αξιολόγησης: Εάν δεν υπάρχει ήδη αξιολόγηση, καλείται η αποθηκευμένη διαδικασία `evaluate_promotion` για τον υπολογισμό της αξιολόγησης και το αποτέλεσμα αποθηκεύεται στην παράμετρο εξόδου `p_result`.
- 7) Τέλος, επιστρέφεται ένα μήνυμα που δηλώνει ότι η αξιολόγηση υπολογίστηκε με επιτυχία, μαζί με το αποτέλεσμα της αξιολόγησης.

κώδικας και παραδείγματα στο Κεφάλαιο 2

Ερώτημα 3.1.3.2

Σε αυτό το [PROCEDURE](#) θα χρειαστούμε πρόσβαση στον πίνακα `evaluation_assignments` για αν ελέγξουμε σε ποια δουλειά έχει ανατεθεί ο κάθε βαθμολογητής. Ακόμη θέλουμε να μπορούμε να ελέγξουμε την εταιρία που δουλεύει ο κάθε βαθμολογητής μέσω του πίνακα `evaluator` και τέλος, θέλουμε πρόσβαση στα στοιχεία κάθε αίτησης μέσω του πίνακα `application_details`. Υλοποιήθηκε, έτσι, η `stored ManageApplication` η οποία λειτουργεί ως εξής:

- 1) Ορισμός παραμέτρων (`p_employee_username`, `p_job_id`, `p_action`)
- 2) Δήλωση μεταβλητών `v_evaluator1_username` και `v_evaluator2_username` που χρησιμοποιούνται για την αποθήκευση ονομάτων αξιολογητών, `v_application_status` που χρησιμοποιείται για την αποθήκευση της κατάστασης της αίτησης εργασίας και `v_same_firm` που χρησιμοποιείται για τον προσδιορισμό της εταιρείας των αξιολογητών
- 3) Ελέγχεται εάν η παράμετρος `p_action` είναι έγκυρη ('i', 'c' ή 'a'). Σε διαφορετική περίπτωση, εμφανίζεται σφάλμα με το αντίστοιχο μήνυμα.
- 4) Ελέγχεται εάν υπάρχει ο υπάλληλος με το συγκεκριμένο όνομα χρήστη (`p_employee_username`) και εάν υπάρχει η θέση εργασίας με το συγκεκριμένο `id` (`p_job_id`). Σε περίπτωση που δεν υπάρχουν, εμφανίζονται αντίστοιχα σφάλματα.
- 5) Λαμβάνονται τα ονόματα των `evaluators` από τον αντίστοιχο πίνακα για την συγκεκριμένη εργασία
- 6) Εάν δεν έχουν συμπληρωθεί οι αξιολογητές τότε συμπληρώνονται μέσω ενός ελέγχου που αναθέτει βαθμολογητές από την ίδια εταιρία
- 7) Ανάλογα με την κατάσταση της αίτησης (`v_application_status`) και τη δράση (`p_action`), πραγματοποιούνται οι κατάλληλες ενέργειες. Για παράδειγμα, εισαγωγή νέας αίτησης ('i'), ακύρωση ενεργής αίτησης ('c'), ή ενεργοποίηση ακυρωμένης αίτησης ('a').

κώδικας και παραδείγματα στο Κεφάλαιο 2

Ερώτημα 3.1.3.3

Όπως έχουμε προαναφέρει στην Απαίτηση 3.1.2.2, φτιάξαμε ένα **PROCEDURE** `evaluate_promotion` για να χρησιμοποιηθεί σε αυτό το ερώτημα. Επίσης κατασκευάσαμε 2 νέους πίνακες τον `employee_jobs` και `job_positions` που θα χρειαστούν εντός του νέου **PROCEDURE** `fill_job_position`, σύμφωνα με τον παρακάτω κώδικα:

```
CREATE TABLE IF NOT EXISTS employee_jobs (  
    id INT(11) AUTO_INCREMENT NOT NULL,  
    job_id INT(11) NOT NULL,  
    employee_username VARCHAR(30) NOT NULL,  
    rating DECIMAL(5,2) NOT NULL,  
    selection_date DATETIME DEFAULT CURRENT_TIMESTAMP,  
    PRIMARY KEY (id),  
    CONSTRAINT emp_jobs_job FOREIGN KEY (job_id) REFERENCES job(id) ON DELETE CASCADE ON  
    UPDATE CASCADE,  
    CONSTRAINT emp_jobs_employee FOREIGN KEY (employee_username) REFERENCES  
    employee(username) ON DELETE CASCADE ON UPDATE CASCADE  
);
```

```
CREATE TABLE IF NOT EXISTS job_positions (  
    job_id INT(11) AUTO_INCREMENT NOT NULL,  
    job_title VARCHAR(255) NOT NULL,  
    status VARCHAR(20) DEFAULT 'open',  
    PRIMARY KEY (job_id)  
);
```

Οι απαιτήσεις είναι φανερές στον κώδικα της `evaluate_promotion` στο ερώτημα 3.1.2.2 και μένει μόνο να καλύψουμε την τελευταία απαίτηση η οποία είναι «Την θέση εργασίας θα παίρνει ο εργαζόμενος με τον υψηλότερο μέσο όρο των δυο αξιολογήσεων. Σε περίπτωση ισοβαθμίας, επιλέγεται αυτός που έκανε νωρίτερα αίτηση».

Αυτό, λοιπόν, επιτυγχάνεται μέσω της διαδικασίας η οποία λειτουργεί ως εξής:

- 1) Ορισμός παραμέτρου p_job_id
- 2) Δήλωση μεταβλητών candidate_username, app_status, evaluator1_username και evaluator2_username, evaluation_result
- 3) Ελέγχεται εάν η θέση εργασίας υπάρχει στους πίνακες αιτήσεων. Σε περίπτωση που δεν υπάρχει, εμφανίζεται ένα σφάλμα.
- 4) Ελέγχεται εάν η θέση εργασίας έχει ήδη παραχωρηθεί. Σε περίπτωση που έχει παραχωρηθεί, εμφανίζεται ένα σφάλμα.
- 5) Χρησιμοποιείται μια **SELECT** για να ληφθεί το όνομα χρήστη του υποψηφίου με την υψηλότερη βαθμολογία για την συγκεκριμένη θέση εργασίας.
- 6) Καλείται η διαδικασία check_application_status για να ελέγξει την κατάσταση της αίτησης του επιλεγμένου υποψηφίου.
- 7) Ελέγχεται αν η αίτηση είναι ακυρωμένη ή έχει ήδη ολοκληρωθεί. Εάν οι αξιολογητές δεν έχουν οριστεί για τη συγκεκριμένη θέση εργασίας, εμφανίζεται ένα σφάλμα.
- 8) Ελέγχεται αν υπάρχει ήδη αξιολόγηση για τον συγκεκριμένο υποψήφιο, αξιολογητές και θέση εργασίας. Σε περίπτωση που δεν υπάρχει, εμφανίζεται ένα σφάλμα.
- 9) Καλείται η διαδικασία evaluate_promotion για να υπολογίσει τη βαθμολογία του υποψηφίου, ανάλογα με τις απαιτήσεις 3.1.2.2.
- 10) Ενημερώνεται η κατάσταση της θέσης εργασίας στον πίνακα job_positions ως "filled".
- 11) Εισάγεται η εγγραφή του υποψηφίου που επιλέγεται στον πίνακα employee_jobs, δηλώνοντας ότι ανατέθηκε στη συγκεκριμένη θέση εργασίας.
- 12) Χρησιμοποιείται μια **SELECT** για να ληφθούν οι τελικές βαθμολογίες του υποψηφίου από τους δύο αξιολογητές.
- 13) Εμφανίζεται ένα μήνυμα επιτυχίας με τις σχετικές πληροφορίες, όπως το id της θέσης εργασίας, το όνομα του υποψηφίου, οι βαθμολογίες των αξιολογητών και η τελική βαθμολογία.

κώδικας και παραδείγματα στο Κεφάλαιο 2

Ερώτημα 3.1.3.4

Έχουμε ήδη δημιουργήσει την **PROCEDURE** για την εισαγωγή των 60.000 εγγραφών, οπότε μπορούμε να φτιάξουμε τα ευρετήρια με δύο πολύ απλές εντολές **CREATE**.

Για το (α): **CREATE INDEX idx_final_rating ON historical_applications (final_rating);**

Για το (β): CREATE INDEX idx_evaluator_filter ON historical_applications (evaluator1_username, evaluator2_username);

Οπότε το μόνο που απομένει είναι να δημιουργήσουμε 2 νέες διαδικασίες, η πρώτη θα εμφανίζει όλες τις αιτήσεις με βαθμό μεταξύ των δύο τιμών που θα εισάγουμε, καθώς και τον εργαζόμενο και το job_id και η δεύτερη θα εμφανίζει όλες τις αιτήσεις που αξιολογήθηκαν από έναν αξιολογητή.

Το (α) υλοποιήθηκε με την Stored **PROCEDURE** filter_applications_by_ratings και λειτουργεί ως εξής:

- 1) Ορισμός παραμέτρων min_rating, max_rating
- 2) Η διαδικασία χρησιμοποιεί μια **SELECT** για να επιλέξει τα ονόματα των υποψηφίων (cand_username), τα id των θέσεων εργασίας (job_id), και τις τελικές βαθμολογίες (final_rating) από τον πίνακα historical_applications. Το **USE INDEX** (idx_final_rating) χρησιμοποιείται για να επιλέξει το συγκεκριμένο δείκτη, πιθανώς βελτιώνοντας την απόδοση (θα φανεί με παραδείγματα στο Κεφάλαιο 2).
- 3) Χρησιμοποιείται η **WHERE** για να επιλέξει μόνο αυτές τις εγγραφές των οποίων τελική βαθμολογία (final_rating) βρίσκεται εντός του καθορισμένου εύρους (min_rating έως max_rating).
- 4) Το **ORDER BY final_rating** είναι προαιρετικό και χρησιμοποιείται για να ταξινομήσει τα αποτελέσματα με βάση την τελική βαθμολογία.

Το (β) υλοποιήθηκε με την Stored **PROCEDURE** filter_applications_by_evaluator και λειτουργεί ως εξής:

- 1) Ορισμός παραμέτρου evaluator_username.
- 2) Χρησιμοποιείται μια **SELECT** για να επιλέξει τα ονόματα των υποψηφίων (cand_username) και τα id των θέσεων εργασίας (job_id) από τον πίνακα historical_applications. Το **USE INDEX** (idx_evaluator_filter) χρησιμοποιείται για να επιλέξει τον συγκεκριμένο δείκτη, πιθανώς βελτιώνοντας την απόδοση του ερωτήματος (θα φανεί με παραδείγματα στο Κεφάλαιο 2).
- 3) Η **WHERE** χρησιμοποιείται για να επιλέξει μόνο αυτές τις εγγραφές στις οποίες ο αξιολογητής 1 (evaluator1_username) ή ο αξιολογητής 2 (evaluator2_username) ισούται με το καθορισμένο evaluator_username.

κώδικας και παραδείγματα στο Κεφάλαιο 2

Ερώτημα 3.1.4.1

Καλούμαστε σε αυτό το σημείο να δημιουργήσουμε ένα trigger που θα καταγράφει κάθε ενέργεια των admins σε έναν πίνακα log.

Οπότε κατασκευάζουμε τον πίνακα dba_log ως εξής:

```
CREATE TABLE IF NOT EXISTS dba_log (  
  log_id INT(11) AUTO_INCREMENT NOT NULL,  
  dba_id INT(11) NOT NULL,  
  username VARCHAR(30) NOT NULL,  
  table_name VARCHAR(30) NOT NULL,  
  action_description VARCHAR(255) NOT NULL,  
  action_date DATETIME NOT NULL,  
  PRIMARY KEY (log_id),  
  FOREIGN KEY (dba_id) REFERENCES dba(id) ON DELETE CASCADE ON UPDATE CASCADE  
);
```

Επόμενο βήμα είναι η δημιουργία 3 triggers για κάθε πίνακα (job,user,degree), η δομή τους θα είναι σχεδόν πανομοιότυπη με μόνη διαφορά κάθε φορά την ενέργεια και τον πίνακα στον οποίο γίνεται αυτή η ενέργεια. Στην ουσία, λέμε στο πρόγραμμα μας να εισάγει στον πίνακα dba_log μία νέα εγγραφή και τα **VALUES** παίρνουν τις τιμές που αναλογούν.

κώδικας και παραδείγματα στο Κεφάλαιο 3

Ερώτημα 3.1.4.2

Το ζητούμενο αυτό έρχεται σε συνεργασία με την Απαίτηση 3.1.2.1. Έχουμε ήδη δημιουργήσει τον πίνακα application_details στον οποίο περνάμε τα στοιχεία κάθε αίτησης. Θα χρησιμοποιηθεί εντός του trigger για να ελέγχουμε αν τηρούνται οι προϋποθέσεις για την ολοκλήρωση μιας εντολής **INSERT**. Ο trigger before_insert_application θα ασχολείται με την εισαγωγή αιτήσεων και λειτουργεί ως εξής:

- 1) Ορισμός μεταβλητών active_applications_count, job_start_date
- 2) Υπολογίζει τον αριθμό των ενεργών αιτήσεων για τον υπάλληλο που υποβάλλει τη νέα αίτηση και χρησιμοποιεί τη μεταβλητή active_applications_count για να αποθηκεύσει το αποτέλεσμα.
- 3) Ελέγχει εάν ο αριθμός των ενεργών αιτήσεων είναι ήδη 3 ή περισσότερες. Εάν ισχύει, εμφανίζεται ένα σφάλμα που δηλώνει ότι ένας υπάλληλος δεν μπορεί να έχει περισσότερες από τρεις ενεργές αιτήσεις.
- 4) Ελέγχει εάν η ημερομηνία αίτησης είναι εντός 15 ημερών πριν από την ημερομηνία έναρξης της θέσης εργασίας. Εάν η ημερομηνία αίτησης είναι μέσα σε αυτό το διάστημα, εμφανίζεται ένα σφάλμα που δηλώνει ότι οι νέες αιτήσεις

πρέπει να υποβάλλονται τουλάχιστον 15 ημέρες πριν από την ημερομηνία έναρξης.

κώδικας και παραδείγματα στο Κεφάλαιο 3

Ερώτημα 3.1.4.3

Ομοίως με το παραπάνω ερώτημα, θα βασιστούμε στις απαιτήσεις του 3.1.2.1 και θα κάνουμε χρήση του πίνακα `application_details` και θα ακολουθήσουμε μια παρόμοια δόμη στο `trigger` μας. Ο `trigger before_update_application` θα ασχολείται με την τροποποίηση αιτήσεων και λειτουργεί ως εξής:

- 1) Ορισμός μεταβλητών `active_applications_count`, `job_start_date`.
- 2) Υπολογίζει τον αριθμό των ενεργών αιτήσεων για τον υπάλληλο που υποβάλλει την ενημέρωση και χρησιμοποιεί τη μεταβλητή `active_applications_count` για να αποθηκεύσει το αποτέλεσμα.
- 3) Ελέγχει εάν ο αριθμός των ενεργών αιτήσεων είναι ήδη 3 ή περισσότερες. Εάν ισχύει, εμφανίζεται ένα σφάλμα που δηλώνει ότι ένας υπάλληλος δεν μπορεί να έχει περισσότερες από τρεις ενεργές αιτήσεις.
- 4) Εάν η ενημέρωση αφορά μια ακύρωση αίτησης (`status` γίνεται `'cancelled'`), τότε εκτελούνται επιπλέον ελέγχοι.
- 5) Έλεγχος εάν η ημερομηνία αίτησης είναι λιγότερο από 10 ημέρες πριν από την ημερομηνία έναρξης της θέσης εργασίας. Εάν ισχύει, εμφανίζεται ένα σφάλμα που δηλώνει ότι μια αίτηση δεν μπορεί να ακυρωθεί εντός 10 ημερών πριν από την ημερομηνία έναρξης.
- 6) Εάν η ενημέρωση αφορά την επαναφορά (`'cancelled'` γίνεται `'active'`) μιας ακυρωμένης αίτησης, τότε ελέγχει εάν ο αριθμός των ενεργών αιτήσεων είναι ήδη 3 ή περισσότερες. Εάν ισχύει, εμφανίζεται ένα σφάλμα που δηλώνει ότι ένας υπάλληλος δεν μπορεί να έχει περισσότερες από τρεις ενεργές αιτήσεις.

κώδικας και παραδείγματα στο Κεφάλαιο 3

Κεφάλαιο 2

Ερώτημα 3.1.3.1

ΚΩΔΙΚΑΣ

```
DELIMITER //
```

```
CREATE PROCEDURE CheckEvaluation(  
    IN p_evaluator_username VARCHAR(255),  
    IN p_employee_username VARCHAR(255),  
    IN p_job_id INT,  
    OUT p_result INT  
)  
BEGIN  
    DECLARE message_text VARCHAR(255); -- Declare a variable to hold the message  
  
    -- Check if the evaluator exists for the specified job_id IN evaluation_assignments  
    SELECT COUNT(*)  
    INTO p_result  
    FROM evaluation_assignments  
    WHERE evaluator1_username = p_evaluator_username OR evaluator2_username = p_evaluator_username  
    AND job_id = p_job_id;  
  
    -- If the evaluator does not exist for the specified job_id, set the result to 0  
    IF p_result = 0 THEN  
        SET p_result = 0;  
        SIGNAL SQLSTATE '45000'  
        SET message_text = 'Evaluator not assigned to the specified job.';  
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = message_text;  
    ELSE  
        -- Check if the employee exists  
        SELECT COUNT(*)
```

```

    INTO p_result
  FROM employee
  WHERE username = p_employee_username;

  -- If the employee does not exist, raise an error
  IF p_result = 0 THEN
    SIGNAL SQLSTATE '45000'
    SET message_text = 'Invalid employee username.';
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = message_text;

  ELSE
    -- Check if the evaluation exists for the specified evaluator, employee, and job
    SELECT final_rating
    INTO p_result
    FROM historical_applications
    WHERE (evaluator1_username = p_evaluator_username OR evaluator2_username =
p_evaluator_username)
      AND cand_username = p_employee_username
      AND job_id = p_job_id
    LIMIT 1;

    -- If an evaluation exists, print a message and return the rating
    IF p_result IS NOT NULL THEN
      SET message_text = CONCAT('Evaluation already exists. Rating: ', p_result);
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = message_text;
    ELSE
      -- Calculate the rating using evaluate_promotion PROCEDURE
      CALL evaluate_promotion(p_evaluator_username, p_employee_username, p_job_id, p_result);
      -- Print a message indicating that the evaluation was calculated
      SELECT 'Evaluation calculated using evaluate_promotion' AS message;

    END IF;
  END IF;
END IF;
END //
DELIMITER ;

```

ΠΑΡΑΔΕΙΓΜΑΤΑ ΚΛΗΣΗΣ ΤΗΣ STORED

```
CALL CheckEvaluation('nonexistent_evaluator', 'employee1', 123, @result);
```

❌ 369 20:16:21 CALL CheckEvaluation('nonexistent_evaluator', 'employee1', 123, @result)	Error Code: 1644. Evaluator not assigned to the specified job.
---	--

```
CALL CheckEvaluation('Douvris', 'nonexistent_employee', 123, @result);
```

❌ 371 20:17:46 CALL CheckEvaluation('Douvris', 'nonexistent_employee', 123, @result)	Error Code: 1644. Invalid employee username.
--	--

```
CALL CheckEvaluation('Douvris', 'Lious', 1, @result);
```

❌ 372 20:21:21 CALL CheckEvaluation('Douvris', 'Lious', 1, @result)	Error Code: 1644. Evaluation already exists. Rating: 18
---	---

Σε περίπτωση που κληθεί με σωστούς όρους η CheckEvaluation τότε εκτυπώνεται μήνυμα:

'Evaluation calculated using evaluate_promotion'

Ερώτημα 3.1.3.2

ΚΩΔΙΚΑΣ

DELIMITER //

```
CREATE PROCEDURE ManageApplication(  
    IN p_employee_username VARCHAR(30),  
    IN p_job_id INT,  
    IN p_action CHAR(1)  
)
```

```
BEGIN
```

```
    DECLARE v_evaluator1_username VARCHAR(30);
```

```
    DECLARE v_evaluator2_username VARCHAR(30);
```

```
    DECLARE v_application_status VARCHAR(20);
```

```
-- Check if the action is valid ('i', 'c', or 'a')
```

```
IF NOT (p_action IN ('i', 'c', 'a')) THEN
```

```
    SIGNAL SQLSTATE '45000'
```

```
    SET MESSAGE_TEXT = 'Invalid action. Use "i" for insert, "c" for cancel, or "a" for activate.';
```

```
END IF;
```

```
-- Check if the employee exists
IF NOT EXISTS (SELECT 1 FROM employee WHERE username = p_employee_username) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Invalid employee username.';
END IF;
```

```
-- Check if the job exists
IF NOT EXISTS (SELECT 1 FROM job WHERE id = p_job_id) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Invalid job ID.';
END IF;
```

```
-- Get the evaluators for the job
SELECT evaluator1_username, evaluator2_username
INTO v_evaluator1_username, v_evaluator2_username
FROM evaluation_assignments
WHERE job_id = p_job_id
LIMIT 1;
```

```
-- If evaluators are not assigned, fill them in from the same company
IF v_evaluator1_username IS NULL THEN
    SELECT username
    INTO v_evaluator1_username
    FROM evaluator
    WHERE firm = (SELECT firm FROM evaluator WHERE username = p_employee_username)
    AND username <> p_employee_username
    LIMIT 1;
END IF;
```

```
IF v_evaluator2_username IS NULL THEN
    SELECT username
    INTO v_evaluator2_username
    FROM evaluator
    WHERE firm = (SELECT firm FROM evaluator WHERE username = p_employee_username)
    AND username <> p_employee_username
    AND username <> v_evaluator1_username
```

```

        LIMIT 1;
    END IF;

    -- Call the PROCEDURE to check the application status
    CALL check_application_status(p_employee_username, p_job_id, v_application_status);

    -- Perform actions based on the specified action
    CASE v_application_status
        WHEN 'not_found' THEN
            -- INSERT an application
            IF p_action = 'i' THEN
                INSERT INTO application_details (cand_username, job_id, status)
                VALUES (p_employee_username, p_job_id, 'active');
            END IF;
        WHEN 'active' THEN
            -- Cancel an application
            IF p_action = 'c' THEN
                UPDATE application_details
                SET status = 'cancelled'
                WHERE cand_username = p_employee_username AND job_id = p_job_id;
            END IF;
        WHEN 'cancelled' THEN
            -- Activate a canceled application
            IF p_action = 'a' THEN
                UPDATE application_details
                SET status = 'active'
                WHERE cand_username = p_employee_username AND job_id = p_job_id;
            END IF;
        END CASE;
    END //

DELIMITER ;

```

ΠΑΡΑΔΕΙΓΜΑΤΑ ΚΛΗΣΗΣ ΤΗΣ STORED

```
CALL ManageApplication('Xristopoulos', 1, 'i');
CALL ManageApplication('Xristopoulos', 1, 'c');
CALL ManageApplication('Xristopoulos', 1, 'a');
CALL ManageApplication('Xristopoulos', 1, 'x');
CALL ManageApplication('akuros', 1, 'i');
CALL ManageApplication('Xristopoulos', 999, 'i');
```

✓	241	18:11:28	CALL ManageApplication('Xristopoulos', 1, 'i')	1 row(s) affected
✓	242	18:11:28	CALL ManageApplication('Xristopoulos', 1, 'c')	1 row(s) affected
✓	243	18:11:28	CALL ManageApplication('Xristopoulos', 1, 'a')	1 row(s) affected
✗	244	18:11:28	CALL ManageApplication('Xristopoulos', 1, 'x')	Error Code: 1644. Invalid action. Use 'i' for insert, 'c' for cancel, or 'a' for activate.
✗	245	18:11:28	CALL ManageApplication('akuros', 1, 'i')	Error Code: 1644. Invalid employee username.
✗	246	18:11:28	CALL ManageApplication('Xristopoulos', 999, 'i')	Error Code: 1644. Invalid job ID.

Ερώτημα 3.1.3.3

ΚΩΔΙΚΑΣ

DELIMITER //

```
CREATE PROCEDURE fill_job_position(
    IN p_job_id INT
)
BEGIN
    DECLARE candidate_username VARCHAR(30);
    DECLARE app_status VARCHAR(20);
    DECLARE evaluator1_username VARCHAR(30);
    DECLARE evaluator2_username VARCHAR(30);
    DECLARE evaluation_result INT;

    -- Check if the job_id exists in the applications
    DECLARE job_exists INT;
    SELECT COUNT(*) INTO job_exists
    FROM application_details
    WHERE job_id = p_job_id;
```

```

-- If the job_id does not exist, print an error message and exit the procedure
IF job_exists = 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Job does not exist in applications. Cannot fill position.';
ELSE
    -- Check if the job is already occupied
    SELECT COUNT(*) INTO @job_occupied
    FROM employee_jobs
    WHERE job_id = p_job_id;

    IF @job_occupied > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Job is already occupied. Cannot fill position.';
    ELSE
        -- Get the candidate's username from historical_applications
        SELECT cand_username INTO candidate_username
        FROM historical_applications
        WHERE job_id = p_job_id
        ORDER BY final_rating DESC
        LIMIT 1;

        -- Call the procedure to check the application status
        CALL check_application_status(candidate_username, p_job_id, app_status);

        -- Check if the application is cancelled or completed
        IF TRIM(app_status) = 'cancelled' THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Selected candidate has cancelled application. Cannot fill position.';
        ELSEIF TRIM(app_status) = 'completed' THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Selected candidate has already completed the application. Cannot fill
position.';
        ELSE
            -- Check if evaluators are assigned for the given job_id
            SELECT ea.evaluator1_username, ea.evaluator2_username
            INTO evaluator1_username, evaluator2_username
            FROM evaluation_assignments ea

```



```

WHERE ea.job_id = p_job_id
LIMIT 1;

-- If evaluators are not assigned, print an error message and exit the procedure
IF evaluator1_username IS NULL OR evaluator2_username IS NULL THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Evaluators are not assigned for the given job_id. Cannot fill position.';
END IF;

-- Check if the evaluation exists for the specified evaluator, employee, and job
SELECT COALESCE(final_rating, 0)
INTO evaluation_result
FROM historical_applications
WHERE (evaluator1_username = evaluator1_username OR evaluator2_username =
evaluator2_username)
AND cand_usurname = candidate_username
AND job_id = p_job_id
LIMIT 1;

-- If the evaluation does not exist, print an error message and exit the procedure
IF evaluation_result = 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Evaluation does not exist. Cannot fill position.';
ELSE
    -- Call the procedure to evaluate promotion
    CALL evaluate_promotion(candidate_username, p_job_id);

    -- Update the job status as filled
    UPDATE job_positions
    SET status = 'filled'
    WHERE job_id = p_job_id;

    -- Insert the job assignment into the employee_jobs table
INSERT INTO employee_jobs (job_id, employee_username, rating)
SELECT p_job_id, cand_usurname, final_rating
FROM historical_applications
WHERE job_id = p_job_id

```

```
ORDER BY final_rating DESC
```

```
LIMIT 1;
```

```
-- Get the ratings for the selected candidate
```

```
SELECT final_rating, final_rating AS 'Evaluator 1 Rating', final_rating AS 'Evaluator 2 Rating'
```

```
INTO @final_rating, @rating1, @rating2
```

```
FROM historical_applications
```

```
WHERE job_id = p_job_id
```

```
ORDER BY final_rating DESC
```

```
LIMIT 1;
```

```
-- Print a success message with ratings and final evaluation
```

```
SELECT CONCAT(
```

```
    'Job position filled for job_id: ', p_job_id,
```

```
    ' with candidate: ', candidate_username,
```

```
    ' Evaluator 1 Rating: ', @rating1,
```

```
    ' Evaluator 2 Rating: ', @rating2,
```

```
    ' Final Rating: ', @final_rating
```

```
) AS 'Success Message';
```

```
    END IF;
```

```
    END IF;
```

```
    END IF;
```

```
    END IF;
```

```
END //
```

```
DELIMITER ;
```

ΠΑΡΑΔΕΙΓΜΑΤΑ ΚΛΗΣΗΣ ΤΗΣ STORED

```
CALL fill_job_position(0);
```

message
▶ Job does not exist.

```
CALL fill_job_position (7);
```

```
Error Code: 1644 Evaluators are not assigned for the given job_id. Skipping evaluation.
```

```
CALL fill_job_position (5);
```

Success Message
▶ Job position filled for job_id: 5 with candidate: candidate2. Evaluator 1 Rating: 20. Evaluator 2 Rating: 20. Final Rating: 20
Job position filled for job_id: 5 with candidate: candidate2. Evaluator 1 Rating: 20. Evaluator 2 Rating: 20. Final Rating: 20

CALL fill_job_position (5);

606 21:11:35 CALL fill_job_position(5)	Error Code: 1644. Job is already occupied. Cannot fill position.
--	--

Ερώτημα 3.1.3.4

ΚΩΔΙΚΑΣ

-- CREATE the index on final_rating column

CREATE INDEX idx_final_rating ON historical_applications (final_rating);

CREATE INDEX idx_evaluator_filter ON historical_applications (evaluator1_username,
evaluator2_username);

-- a)

DELIMITER //

CREATE PROCEDURE filter_applications_by_ratings(IN min_rating INT, IN max_rating INT)

BEGIN

-- SELECT cand_usurname, job_id, and final_rating based on the rating range

SELECT cand_usurname AS employee_username, job_id, final_rating

FROM historical_applications USE INDEX (idx_final_rating)

WHERE final_rating BETWEEN min_rating AND max_rating;

END //

DELIMITER ;

-- b)

DELIMITER //

CREATE PROCEDURE filter_applications_by_evaluator(IN evaluator_username VARCHAR(30))

BEGIN

-- SELECT cand_usurname (employee username) and job_id for applications evaluated by the specified evaluator

SELECT cand_usurname AS employee_username, job_id

FROM historical_applications USE INDEX (idx_evaluator_filter)

WHERE evaluator1_username = evaluator_username OR evaluator2_username = evaluator_username;

END //

DELIMITER ;

ΠΑΡΑΔΕΙΓΜΑΤΑ ΚΛΗΣΗΣ ΤΗΣ STORED

Α΄ ΠΕΡΙΠΤΩΣΗ

✓ 3085 21:02:14 CALL filter_applications_by_ratings(5, 10) 12470 row(s) returned 0.031 sec / 0.015 sec

Συνολικός χρόνος με index 0.031/ Χωρίς index 0.015

```
1000 • SET PROFILING = 0;
1001 • CALL filter_applications_by_ratings(5, 10);
```

Result Grid | Filter Rows: | Export: | Wrap C

	employee_username	job_id	final_rating
▶	Nikolakopoulou	8	5
	Lious	3	5
	Papanikolaou	1	5
	Xristopoulos	6	5
	Papapaulou	2	5
	Nikolakopoulou	6	5
	Nikolakopoulou	4	5
	Papanikolaou	5	5
	Ioannidh	8	5
	Nikolakopoulou	4	5
	Papapaulou	4	5
	Lious	5	5
	Papapaulou	7	5
	Papanikolaou	2	5

Β΄ ΠΕΡΙΠΤΩΣΗ

✓ 3275 21:22:42 CALL filter_applications_by_evaluator('Douvris') 12780 row(s) returned 0.016 sec / 0.031 sec

Συνολικός χρόνος με index 0.016/ Χωρίς index 0.031

```
1013 • CALL filter_applications_by_evaluator('Douvris');
```

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

	employee_username	job_id
▶	Ioannidh	1
	Ioannidh	6
	Nikolakopoulou	5
	Papapaulou	1
	Papanikolaou	6
	Nikolakopoulou	8
	Lious	7
	Xristopoulos	4
	Ioannidh	5
	Xristopoulos	1
	Papanikolaou	7
	Xristopoulos	1
	Lious	8
	Nikolakopoulou	4

Κεφάλαιο 3

Ερώτημα 3.1.4.1

JOB TABLE

ΚΩΔΙΚΑΣ

DELIMITER //

```
CREATE TRIGGER job_after_insert
AFTER INSERT ON job
FOR EACH ROW
BEGIN
    INSERT INTO dba_log (dba_id, username, action_description, table_name, action_date)
    VALUES (@current_admin_id, @current_admin, 'insert', 'job', NOW());
END;
//
```

```
CREATE TRIGGER job_after_update
AFTER UPDATE ON job
FOR EACH ROW
BEGIN
    INSERT INTO dba_log (dba_id, username, action_description, table_name, action_date)
    VALUES (@current_admin_id, @current_admin, 'update', 'job', NOW());
END;
//
```

```
CREATE TRIGGER job_after_delete
AFTER DELETE ON job
FOR EACH ROW
BEGIN
    INSERT INTO dba_log (dba_id, username, action_description, table_name, action_date)
    VALUES (@current_admin_id, @current_admin, 'delete', 'job', NOW());
END;
//
```

DELIMITER ;

ΠΑΡΑΔΕΙΓΜΑΤΑ ΚΛΗΣΗΣ ΤΟΥ TRIGGER

INSERT INTO job (start_date, salary, position, edra, evaluator, submission_date)

VALUES ('2022-05-01', 75000, 'Data Analyst', 'Analytics Department', 'Douvris', '2022-05-15');

UPDATE job SET salary = 80000 WHERE id = 1;

DELETE FROM job WHERE id = 2;

SELECT * FROM dba_log;

	log_id	dba_id	username	table_name	action_description	action_date
▶	1	1	theokatsa	job	insert	2024-01-15 23:04:57
	2	1	theokatsa	job	update	2024-01-15 23:04:57
	3	1	theokatsa	job	delete	2024-01-15 23:04:57

USER TABLE

ΚΩΔΙΚΑΣ

DELIMITER //

CREATE TRIGGER user_after_insert

AFTER INSERT ON user

FOR EACH ROW

BEGIN

INSERT INTO dba_log (dba_id, username, action_description, table_name, action_date)

VALUES (@current_admin_id, @current_admin, 'insert', 'user', NOW());

END;

//

CREATE TRIGGER user_after_update

AFTER UPDATE ON user

FOR EACH ROW

BEGIN

INSERT INTO dba_log (dba_id, username, action_description, table_name, action_date)

VALUES (@current_admin_id, @current_admin, 'update', 'user', NOW());

END;

//

CREATE TRIGGER user_after_delete

AFTER DELETE ON user

FOR EACH ROW

BEGIN

```

INSERT INTO dba_log (dba_id, username, action_description, table_name, action_date)
VALUES (@current_admin_id, @current_admin, 'delete', 'user', NOW());
END;
//
DELIMITER ;

```

ΠΑΡΑΔΕΙΓΜΑΤΑ ΚΛΗΣΗΣ ΤΟΥ TRIGGER

```

INSERT INTO user (username, password, name, lastname, email)
VALUES ('new_user', 'password123', 'John', 'Doe', 'new_user@example.com');
UPDATE user SET email = 'updated_email@example.com' WHERE username = 'new_user';
DELETE FROM user WHERE username = 'new_user';
SELECT * FROM dba_log;

```

4	1	theokatsa	user	insert	2024-01-15 23:22:18
5	1	theokatsa	user	update	2024-01-15 23:22:19
6	1	theokatsa	user	delete	2024-01-15 23:22:19

DEGREE TABLE

ΚΩΔΙΚΑΣ

```

DELIMITER //
CREATE TRIGGER degree_after_insert
AFTER INSERT ON degree
FOR EACH ROW
BEGIN
    INSERT INTO dba_log (dba_id, username, action_description, table_name, action_date)
    VALUES (@current_admin_id, @current_admin, 'insert', 'degree', NOW());
END;
//
CREATE TRIGGER degree_after_update
AFTER UPDATE ON degree
FOR EACH ROW
BEGIN
    INSERT INTO dba_log (dba_id, username, action_description, table_name, action_date)

```

```

VALUES (@current_admin_id, @current_admin, 'update', 'degree', NOW());
END;
//
CREATE TRIGGER degree_after_delete
AFTER DELETE ON degree
FOR EACH ROW
BEGIN
    INSERT INTO dba_log (dba_id, username, action_description, table_name, action_date)
    VALUES (@current_admin_id, @current_admin, 'delete', 'degree', NOW());
END;
//
DELIMITER ;

```

ΠΑΡΑΔΕΙΓΜΑΤΑ ΚΛΗΣΗΣ ΤΟΥ TRIGGER

```

INSERT INTO degree (titlos, idryma, bathmida)
VALUES ('Computer Science', 'University of XYZ', 'BSc');
UPDATE degree SET bathmida = 'MSc' WHERE titlos = 'Computer Science' AND idryma = 'University of XYZ';
DELETE FROM degree WHERE titlos = 'Computer Science' AND idryma = 'University of XYZ';
SELECT * FROM dba_log;

```

7	1	theokatsa	degree	insert	2024-01-15 23:22:20
8	1	theokatsa	degree	update	2024-01-15 23:22:20
9	1	theokatsa	degree	delete	2024-01-15 23:22:20

Ερώτημα 3.1.4.2

ΚΩΔΙΚΑΣ

```

DELIMITER //
CREATE TRIGGER before_insert_application
BEFORE INSERT ON application_details
FOR EACH ROW
BEGIN
    DECLARE active_applications_count INT;
    DECLARE job_start_date DATETIME;

```



```

-- Count the number of active applications for the employee
SELECT COUNT(*)
INTO active_applications_count
FROM application_details
WHERE cand_username = NEW.cand_username
    AND status = 'active';

-- Check if the employee already has three active applications
IF active_applications_count >= 3 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'An employee cannot have more than three active applications.';
END IF;

-- Check if the application DATE is less than 15 days before the job start date
SELECT start_DATE
INTO job_start_DATE
FROM job
WHERE id = NEW.job_id;

IF NEW.application_date < DATE_SUB(job_start_date, INTERVAL 15 DAY) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'New applications must be submitted at least 15 days before the job start
DATE.';
END IF;

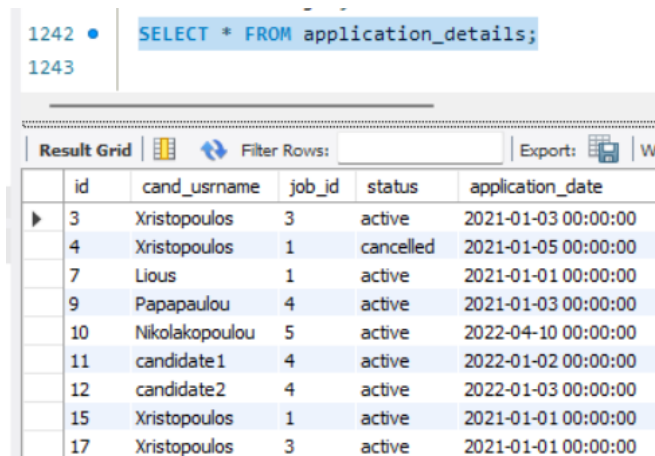
END//
DELIMITER ;

```

ΠΑΡΑΔΕΙΓΜΑΤΑ ΚΛΗΣΗΣ ΤΟΥ TRIGGER

ΟΡΙΟ 3 ΑΙΤΗΣΕΩΝ

Xristopoulos: 3 active
Applications



	id	cand_username	job_id	status	application_date
▶	3	Xristopoulos	3	active	2021-01-03 00:00:00
	4	Xristopoulos	1	cancelled	2021-01-05 00:00:00
	7	Lious	1	active	2021-01-01 00:00:00
	9	Papapaulou	4	active	2021-01-03 00:00:00
	10	Nikolakopoulou	5	active	2022-04-10 00:00:00
	11	candidate1	4	active	2022-01-02 00:00:00
	12	candidate2	4	active	2022-01-03 00:00:00
	15	Xristopoulos	1	active	2021-01-01 00:00:00
	17	Xristopoulos	3	active	2021-01-01 00:00:00

INSERT INTO application_details (cand_username, job_id, status, application_DATE)

VALUES ('Xristopoulos', 1, 'active', '2021-01-01');

✖ 6878 02:30:51 INSERT INTO application_details (cand_username, job_id, status, application_date) VALUES ('Xristopoulos', 1, '... Error Code: 1644. An employee cannot have more than three active applications.

ΟΡΙΟ 15 ΗΜΕΡΩΝ

id	start_date
1	2022-01-01
3	2022-03-01

INSERT INTO application_details (cand_username, job_id, status, application_DATE)

VALUES ('Papapaulou', 3, 'active', '2022-02-15');

INSERT INTO application_details (cand_username, job_id, status, application_DATE)

VALUES ('Papapaulou', 3, 'active', '2022-02-14');

✖ 7157 02:38:44 INSERT INTO application_details (cand_username, job_id, status, application_date) VALUES ('Papapaulou', 3, '... Error Code: 1644. New applications must be submitted at least 15 days before the job start date.

✔ 7158 02:38:54 INSERT INTO application_details (cand_username, job_id, status, application_date) VALUES ('Papapaulou', 3, '... 1 row(s) affected

Ερώτημα 3.1.4.3

ΚΩΔΙΚΑΣ

DELIMITER //

CREATE TRIGGER before_update_application

BEFORE UPDATE ON application_details

FOR EACH ROW

BEGIN

DECLARE active_applications_count INT;

```

DECLARE job_start_date DATETIME;

-- Count the number of active applications for the employee
SELECT COUNT(*)
INTO active_applications_count
FROM application_details
WHERE cand_username = NEW.cand_username
    AND status = 'active';

-- Check if the employee already has three active applications
IF active_applications_count >= 3 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'An employee cannot have more than three active applications.';
END IF;

-- Check if the application is being canceled
IF NEW.status = 'cancelled' THEN
    -- Check if the cancellation date is less than 10 days before the job start date
    SELECT start_date
    INTO job_start_date
    FROM job
    WHERE id = NEW.job_id;

    IF NEW.application_date > DATE_SUB(job_start_date, INTERVAL 10 DAY) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'An application cannot be canceled within 10 days before the job start date.';
    END IF;
END IF;

-- Check if the canceled application is being reactivated and the user has less than 3 active applications
IF NEW.status = 'active' AND active_applications_count >= 3 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'An employee cannot have more than three active applications.';
END IF;

END//

```

DELIMITER ;

ΠΑΡΑΔΕΙΓΜΑΤΑ ΚΛΗΣΗΣ ΤΟΥ TRIGGER

ΟΡΙΟ 10 ΗΜΕΡΩΝ

UPDATE application_details

SET status = 'cancelled', application_date = '2022-02-24'

WHERE cand_usurname = 'Papapaulou' AND job_id = 3 AND status = 'active'; --Start DATE is 2022-03-01

7321 03:00:12 UPDATE application_details SET status = 'cancelled', application_date = '2022-02-24' WHERE cand_usurname... Error Code: 1644. An application cannot be canceled within 10 days before the job start date.

ΟΡΙΟ 3 ΕΝΕΡΓΩΝ ΑΙΤΗΣΕΩΝ

9	Xristopoulos	1	active	2021-01-01 00:00:00
10	Xristopoulos	1	active	2021-01-01 00:00:00
11	Xristopoulos	1	cancelled	2021-01-01 00:00:00
12	Xristopoulos	1	active	2021-01-01 00:00:00

UPDATE application_details

SET status = 'active', application_date = '2021-01-01'

WHERE cand_usurname = 'Xristopoulos' AND job_id = 1 AND status = 'cancelled';

7319 02:57:58 UPDATE application_details SET status = 'active', application_date = '2021-01-01' WHERE cand_usurname = '...' Error Code: 1644. An employee cannot have more than three active applications.

2^ο Μέρος: GUI

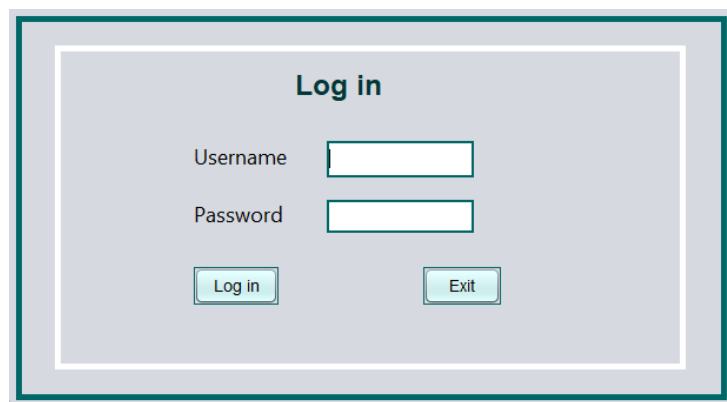
ΚΕΦΑΛΑΙΟ 5

3.2.1

Περιγραφή λειτουργικότητας

Με την χρήση του NetBeans έχουμε κατασκευάσει μία Γραφική Διεπαφή Χρήστη που να κάνει τις παρακάτω λειτουργίες.

Αρχικά εμφανίζεται ένα login page και χρησιμοποιώντας το username από τον πίνακα dba (database admins) και τον κατάλληλο κωδικό μπορεί ο χρήστης να έχει πρόσβαση στην βάση δεδομένων. Με το που συμπληρώσει τα σωστά στοιχεία, εμφανίζεται ένα JFrame με 17 κουμπιά που αντιστοιχούν στους 17 πίνακες της βάσης δεδομένων καθώς και ένα sign out κουμπί που επαναφέρει τον χρήστη στη login page.



Αφού επιλεγεί ένα κουμπί, εμφανίζεται το αντίστοιχο JFrame που περιέχει ένα label στο άνω τμήμα που έχει το όνομα του πίνακα που έχει επιλεγεί. Ακριβώς από κάτω υπάρχει ένα table με το κατάλληλο αριθμό στηλών για τον συγκεκριμένο πίνακα, και μέσα σε αυτόν είναι τα δεδομένα που είχαμε εισάγει στην database στην προπαρασκευαστική φάση. Δίπλα εμφανίζονται 6 κουμπιά και text areas ίδιου αριθμού με τον αριθμό των στηλών. Όταν ο admin επιλέγει μια σειρά από τον πίνακα, στα text areas εμφανίζονται τα περιεχόμενα της γραμμής σε κάθε στήλη. Τα κουμπιά είναι για την έξοδο από την διεπαφή, για την επιστροφή πίσω στο JFrame με την επιλογή των πινάκων, για την εισαγωγή, τροποποίηση, την διαγραφή δεδομένων και για το reset των text areas. Προφανώς σε κάθε JFrame υπάρχει η επιλογή exit από το Gui.

```
private void jobActionPerformed(java.awt.event.ActionEvent evt) {  
    Job jobFrame = new Job();  
    jobFrame.setVisible(true);  
    this.dispose();  
}
```

```
private JFrame frame;  
private void exitActionPerformed(java.awt.event.ActionEvent evt) {  
    frame = new JFrame("Exit");  
    if (JOptionPane.showConfirmDialog(frame, "Are you sure you want to exit?", "Database",  
        JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION) {  
        System.exit(0);  
    }  
}
```

Ένωση με την βάση mysql

Η πρώτη κίνηση που κάναμε με την εκκίνηση της δημιουργίας του GUI ήταν να συνδέσουμε το project με την βάση μας της mysql. Φτιάξαμε μία κλάση με

```
public class sqlconnect {  
    private static final String DB_URL = "jdbc:mysql://localhost:3306/databasefinal" ;  
    String username = "root";  
    String password = "root";  
}
```



το όνομα sqlconnect και προσθέσαμε το package erecruit2023;

Μετά ορίζουμε τις μεταβλητές DB_URL, username και password που καθορίζουν τα στοιχεία σύνδεσης για την βάση δεδομένων. Το URL jdbc:mysql://localhost:3306/databasefinal δείχνει τον τοπικό MySQL server στη θύρα 3306 και τη βάση δεδομένων "databasefinal". Και για να συνδεθεί βάλαμε το username και το password που είναι root και root.

Μετά μέσα στην main βάλαμε try block για την διαχείριση εξαιρέσεων (SQLExceptions). Μέσα σε αυτή δημιουργούμε ένα αντικείμενο Statement για την εκτέλεση SQL εντολών εκτελούμε μια SQL εντολή "SELECT * FROM user" με την μέθοδο executeQuery(), και τα αποτελέσματα αποθηκεύονται στο αντικείμενο ResultSet. Χρησιμοποιούμε την κλάση ResultSetMetaData για να ανακτήσουμε πληροφορίες σχετικά με τη δομή των αποτελεσμάτων, υπολογίζεται το πλήθος των στηλών και εκτυπώνονται τα ονόματα των στηλών.

Τέλος, για τον διαχειρισμό εξαιρέσεων έχουμε βάλει ένα catch που να εκτυπώνει μια στοίβα εξαίρεσης.

```
while (resultSet.next()) {
    for (int i = 1; i <= numberOfColumns; i++)
        System.out.printf("%-8s\t", resultSet.getObject(i));
    System.out.println();
}
} catch (SQLException ex) {
    ex.printStackTrace();
}
}

while (rs.next()) {
    String username = rs.getString("username");
    String password = rs.getString("password");
    String name = rs.getString("name");
    String lastname = rs.getString("lastname");
    String reg_date = rs.getString("reg_date");
    String email = rs.getString("email");

    String tbData[] = {username, password, name, lastname,
        email};
    tblModel.addRow(tbData);
}
```

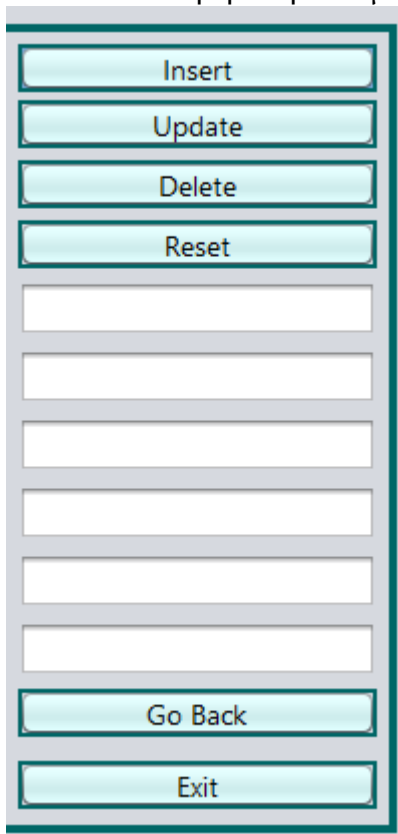
Γενικός σχεδιασμός

Το κύριο JFrame δηλαδή το FirstPage έχει πολύ περιορισμένη λειτουργία. Τα κουμπιά λειτουργικά είναι ίδια, μεταφέρουν τον χρήστη από το FirstPage στα JFrame των πινάκων που ενδιαφέρουν τον διαχειριστή. Το μοντέλο είναι αυτό της διπλανής φωτογραφίας και απλά είναι αναδιαμορφωμένο και για τα 17 κουμπιά. Πιο κάτω είναι η δομή του κουμπιού της εξόδου από το GUI και είναι το ίδιο σε όλα τα jframes. Έχω πιο πάνω ορίσει τις κατάλληλες βιβλιοθήκες ώστε να εμφανίζονται τα JOptionPane για τις αυτές τις μικροεπιλογές. Και ένα παρόμοιο έχουμε δημιουργήσει για το κουμπί sign out αλλά αντί για System.exit(0); Οδηγούμε τον χρήστη στο login page.

```
private static final String username = "root";
private static final String password = "root";
private static final String dataConn = "jdbc:mysql";
Connection sqlConn = null;
PreparedStatement pst = null;
ResultSet rs = null;
int q, i, id, deleteItem;
```

Πιο αναλυτικά τώρα σε όλα τα JFrame με τους πίνακες η δομή είναι ίδια. Τα μόνα Αρχικά ενώνουμε τους πίνακες και δημιουργούμε μεταβλητές για τη σύνδεση (sqlConn), την προετοιμασμένη εντολή (pst) και τα αποτελέσματα (rs) των ερωτημάτων SQL καθώς και 4 έξτρα που θα τους χρησιμοποιήσουμε πιο κάτω.

Κατασκευάζουμε μια μέθοδο `upDateDB()` η οποία θα αναναιώνει τα δεδομένα του πίνακα στην βάση δεδομένων μας. φροντίζουμε τα δώσουμε κατάλληλη εντολή για την



επιλογή του πίνακα (`pst = sqlConn.prepareStatement("select * from user");`). Μετά αφού δημιουργήσουμε πίνακα ρυθμίζουμε τον αριθμό των γραμμών του στο 0 για να τον αδειάσουμε. Ύστερα με χρήση του `while (rs.next())`, επαναφέρουμε τα δεδομένα από το `ResultSet` στον πίνακα `RecordTable` και με την χρήση `for` εμφανίζουμε τις στήλες του πίνακα (οι οποίες διαμορφώνονται ανάλογα με τον πίνακα).

Τέλος, αφού δημιουργήσουμε αυτή την μέθοδο, την καλούμε μέσα στην `public User()`.

CRUD

Πιο κάτω έχουμε αναλύσεις για τα κουμπιά που εκτελούν την διεργασία του `create`, `reset`, `update` και `delete`. Πρώτα για το **update**, δηλαδή για την τροποποίηση των δεδομένων που υπάρχουν ήδη μέσα στην βάση. Καλούμε την κατάλληλη sql εντολή (στην συγκεκριμένη περίπτωση, του `user`, είναι `update user set password=?, name=?, lastname=?, reg_date=?, email=? where username=?`) και ανρίστοιχα έχουμε την `setString` για να παίρνουν τα

αναναιωμένα δεδομένα. Αφού εκτελεστεί αυτή η διεργασία, εμφανίζεται αντίστοιχο μήνυμα (`Update Executed`). Τέλος καλούμε και την `upDateDB()` για να δούμε τα αποτελέσματα στον πίνακα.

Αμέσως μετά έχουμε το **delete**.

Πρώτα ορίζουμε τα `selectedRows`

για το `table` μας και μετά έχουμε ένα `JOptionPane` που ρωτά τον `admin` άμα θέλει όντως να διαγράψει αυτή την σειρά. Συνδέουμε ξανά την βάση και εκτελούμε την κατάλληλη εντολή `delete from user where username=?`. Διαγράφουμε την σειρά, και πάλι με `JOptionPane` εμφανίζεται μήνυμα που να ενημερώνει τον χρήστη ότι όντως διαγράφηκε η σειρά. Μετά με το `setText` βάζουμε όλα τα `textAreas = 0`.

Το τρίτο κουμπί είναι το **reset**. Το φτιάξαμε ώστε να είναι πιο γρήγορη η διαδικασία εισαγωγής δεδομένων, βάζοντας με ένα κουμπί κενό μέσα στα `textAreas`. Βάζουμε το `setText("")` για να κάνει αυτή την δουλειά για εμάς.

Το τελευταίο κουμπί είναι για την

εισαγωγή δεδομένων είναι το **insert**.

Έχουμε κενά τα πεδία μετά το `reset`, οπότε ο χρήστης θα μπορεί να συμπληρώσει τα δεδομένα που θέλει. Οπότε στον κώδικα

```
while(rs.next()){
    Vector columnData = new Vector();
    for (i=1; i<=q; i++){
        columnData.add(rs.getString("username"));
        columnData.add(rs.getString("password"));
        columnData.add(rs.getString("name"));
        columnData.add(rs.getString("lastname"));
        columnData.add(rs.getString("reg_date"));
        columnData.add(rs.getString("email"));
    } RecordTable.addRow(columnData);
}

pst.setString(1, text2.getText());
pst.setString(2, text3.getText());
pst.setString(3, text4.getText());
pst.setString(4, text5.getText());
pst.setString(5, text6.getText());
pst.setString(6, text1.getText());
pst.executeUpdate();
JOptionPane.showMessageDialog(rootPane, "Update Executed");
upDateDB();
}

catch (ClassNotFoundException | SQLException ex) {
    ex.printStackTrace();
}
```


έχουμε το `insert into user(username, password, name, lastname, reg_date, email)values(?, ?, ?, ?, ?, ?)`. Μετά ακολουθούμε τα ίδια βήματα με την update και καλούμε την `updateDB()`. Στο τέλος εμφανίζουμε το αντίστοιχο μήνυμα.

Login Page

Στην σελίδα για την σύνδεση έχουμε 2 text areas και 2 κουμπιά, για την σύνδεση του χρήστη και την έξοδο από την εφαρμογή. Καλούμε από τον πίνακα dba τα username των διαχειριστών και τα password. Με το που πατήσουν το κουμπί του login οι χρήστες έχουμε ένα try. Σε περίπτωση που είναι λάθος ο συνδιασμός κάνει pop up το αντίστοιχο μήνυμα και τα textareas έχουν και τα δύο κενό. Αν είναι σωστός, ο κώδικας οδηγεί τον admin στην FirstPage. Όλο αυτό γίνεται με την χρήση του if statement.

```
String sql = "Select * from dba where username =? and id =?";
PreparedStatement pst = con.prepareStatement(sql);
pst.setString(1, usernameText.getText());
pst.setString(2, passwordText.getText());
ResultSet rs = pst.executeQuery();
```

3.2.2. Λεπτομέρειες στα JFrames

Είναι πιθανόν κατά την εισαγωγή δεδομένων σε πίνακες ο χρήστης να μην πληκτρολογήσει ένα όνομα σωστά. Για να αποφύγουμε τέτοιου είδους λάθη, αντικαταστήσαμε ορισμένα text Areas με combo box. Για παράδειγμα αυτά που δίνουν ονόματα evaluators στο job ή που περιγράφουν την φάση των αιτήσεων (active, cancelled, complete).

Ο τρόπος με τον οποίο δημιουργήσαμε αυτά τα combo box είναι παρόμοιος στα jframes μεταβάλλοντας μόνο τα ονόματα πινάκων και στηλών. Πιο κάτω αναφέρεται ένα μοτίβο που ακολουθήσαμε για την αλλαγή ενός text area σε combo box πάνω στο Project frame.

Αρχικά ορίσαμε `populateEmployeeBox()` (αλλάζει το όνομα ανάλογα με το jframe και την στήλη που θέλουμε να αλλάξουμε). Συνδέουμε αυτή τη μέθοδο με την βάση, και μετά καλούμε την στήλη από τον πίνακα που θέλουμε. Τα δεδομένα που παίρνουμε, μέσω while loop, περνιούνται στο combo box.

```
private void populateEmployeeBox() {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/erecruit2023?useSSL=false",
            "root", "root");
        Statement st = con.createStatement();
        String sql = "SELECT username FROM employee";
        ResultSet rs = st.executeQuery(sql);
        DefaultComboBoxModel<String> model = new DefaultComboBoxModel<>();
        while (rs.next()) {
```

```
String username = rs.getString("username");
model.addElement(username);}

employeeBox.setModel(model);

con.close();
} catch (Exception e) {
    e.printStackTrace();}}
```

Μετά την καλούμε μέσα στην project ώστε να εφαρμοστεί αυτό που φτιάξαμε. Και μετά πάμε σε κάθε κουμπί του CRUD και αντικαθιστούμε το κουμπί που είχαμε με το combo box.

Πιο συγκεκριμένα, στο update button, η μορφή θα μεταβληθεί κάπως έτσι.

```
pst.setString(1, text2.getText());
pst.setString(2, text3.getText());
pst.setString(3, text4.getText());
//pst.setString(4, text1.getText());
pst.setString(4, (String)
employeeBox.getSelectedItems());
```

```
private String getSelectedLanguages() {
    StringBuilder selectedLanguages = new StringBuilder();
    if (GR.isSelected()) selectedLanguages.append("GR ");
    if (EN.isSelected()) selectedLanguages.append("EN ");
    if (FR.isSelected()) selectedLanguages.append("FR ");
    if (GE.isSelected()) selectedLanguages.append("GE ");
    if (SP.isSelected()) selectedLanguages.append("SP ");
    if (CH.isSelected()) selectedLanguages.append("CH ");
    return selectedLanguages.toString().trim();
}
```

candid	lang
Galanh	CH
Liou	EN
Nikolakopoulou	SP
Papapaulou	CH
Xristopoulos	EN

Και στην λείτουργία μου φτιάξαμε tableMouseClicked κάναμε το ακόλουθο:

```
// text1.setText(RecordTable.getValueAt(SelectedRows,0).toString());
employeeBox.setSelectedItem(RecordTable.getValueAt(SelectedRows, 0).toString());
```

Τις παραπάνω αλλαγές τις κάναμε όπου θεωρήσαμε ότι ήταν απαραίτητο. Μερικοί από τους πίνακες που προσθέσαμε combo box είναι: languages, project, job, has_degree, application_details κτλπ.

Επιπλέον, θεωρήσαμε πιο βολικό για τον χρήστη να τροποποιήσουμε τον τρόπο που επιλέγονται οι γλώσσες στον πίνακα languages. Πιο αναλυτικά, χρησιμοποιήσαμε 6 check boxies με τις 6 γλώσσες που μας ενδιαφέρουν. Δημιουργούμε μια μέθοδο getSelectedLanguages() η οποία στην αρχή δημιουργεί ένα αντικείμενο StringBuilder για να δημιουργήσει το string που θα χρησιμοποιήσουμε. Μετά με την μέθοδο selectedLanguages ελέγχονται ποια κουτιά είναι τσεκαρισμένα, και προσθέτονται στο StringBuilder. Το τελικό αποτέλεσμα προκύπτει από την μετατροπή του StringBuilder σε ένα string χρησιμοποιώντας την μέθοδο toString(), και με την μέθοδο trim() αφαιρούνται οποιαδήποτε κενά υπάρχουν. Η μέθοδο στο τέλος επιστρέφει το string που περιέχει τις γλώσσες που θέλουμε.

```
public Project() {
    initComponents();
    displayprojectData();
    populateEmployeeBox();
}
```

Επιπρόσθετα, χρησιμοποιήσαμε και άλλα combo box , τα οποία περιέχουν είτε χρονολογίες είτε καταστάσεις (active, cancelled, completed). Θεωρήσαμε ότι ήταν απαραίτητα να προστεθούν καθώς κάνουν το GUI πιο εύκολο στην χρήση του αποφεύγοντας πιθανά λάθη.

ΑΚΟΛΟΥΘΕΙ ΤΟ ΣΥΝΟΛΟ ΤΟΥ ΚΩΔΙΚΑ ΤΗΣ SQL

```
-- ΠΡΟΠΑΡΑΣΚΕΥΑΣΤΙΚΟ ΣΤΑΔΙΟ-ΕΡΩΤΗΜΑ 3.1.1 --
```

```
DROP DATABASE if exists erecruit2023;
```

```
CREATE DATABASE erecruit2023;
```

```
USE erecruit2023;
```

```
CREATE TABLE IF NOT EXISTS etaireia (
```

```
    AFM CHAR(9) DEFAULT 'unknown' NOT NULL,
```

```
    DOY VARCHAR(30) DEFAULT 'unknown' NOT NULL,
```

```
    name VARCHAR(35) DEFAULT 'unknown' NOT NULL,
```

```
    tel VARCHAR(10) DEFAULT '0' NOT NULL,
```

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

```
    num INT(11) DEFAULT '0' NOT NULL,
```

```
    city VARCHAR(45) DEFAULT 'unknown' NOT NULL,
```

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

```
    PRIMARY KEY (AFM)
```

```
);
```

```
CREATE TABLE IF NOT EXISTS user(
```

```
    username VARCHAR(30) DEFAULT 'unknown' NOT NULL,
```

```
    password VARCHAR(20) DEFAULT 'unknown' NOT NULL,
```

```
    name VARCHAR(25) DEFAULT 'unknown' NOT NULL,
```

```
    lastname VARCHAR(35) DEFAULT 'unknown' NOT NULL,
```

```
    reg_date DATETIME DEFAULT CURRENT_TIMESTAMP,
```

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

```
    PRIMARY KEY(username)
```

```
);
```

```
CREATE TABLE IF NOT EXISTS evaluator(
```

```
    username VARCHAR(30) DEFAULT 'unknown' NOT NULL,
```

```

        exp_years TINYINT(4) DEFAULT '0' NOT NULL,
        firm CHAR(9) DEFAULT 'unknown' NOT NULL,
        PRIMARY KEY(username),
        CONSTRAINT eval_etaireia FOREIGN KEY (firm) REFERENCES etaireia(AFM)
        ON DELETE CASCADE ON UPDATE CASCADE,
        CONSTRAINT eval_username FOREIGN KEY (username) REFERENCES user(username)
        ON DELETE CASCADE ON UPDATE CASCADE
    );

```

```

CREATE TABLE IF NOT EXISTS subject(
    title VARCHAR(36) DEFAULT 'unknown' NOT NULL,
    descr TINYTEXT NOT NULL,
    belongs_to VARCHAR(36) DEFAULT NULL,
    PRIMARY KEY(title),
        CONSTRAINT subjbelongs FOREIGN KEY (belongs_to) REFERENCES subject(title)
        ON DELETE CASCADE ON UPDATE CASCADE
);

```

```

CREATE TABLE IF NOT EXISTS job (
    id INT(11) AUTO_INCREMENT NOT NULL,
    start_date DATE NOT NULL,
    salary FLOAT DEFAULT 0 NOT NULL,
    position VARCHAR(60) DEFAULT 'unknown' NOT NULL,
    edra VARCHAR(60) DEFAULT 'unknown' NOT NULL,
    evaluator VARCHAR(30) DEFAULT 'unknown' NOT NULL,
    announce_date DATETIME DEFAULT current_timestamp(),
    submission_date DATE NOT NULL ,
    PRIMARY KEY(id),
    CONSTRAINT eval_job FOREIGN KEY (evaluator) REFERENCES evaluator(username)
    ON DELETE CASCADE ON UPDATE CASCADE
);

```

```

CREATE TABLE IF NOT EXISTS requires (
    job_id INT(11) DEFAULT '0' NOT NULL,
    subject_title VARCHAR(36) DEFAULT 'unknown' NOT NULL,
    PRIMARY KEY (job_id, subject_title),
    CONSTRAINT requires_job FOREIGN KEY (job_id) REFERENCES job(id)

```

```
ON DELETE CASCADE ON UPDATE CASCADE  
);
```

```
CREATE TABLE IF NOT EXISTS employee(  
    username VARCHAR(30) DEFAULT 'unknown' NOT NULL,  
    bio TEXT NOT NULL,  
    sistatikes VARCHAR(35) DEFAULT 'unknown' NOT NULL,  
    certificates VARCHAR(35) DEFAULT 'unknown' NOT NULL,  
    PRIMARY KEY(username),  
    CONSTRAINT empl_username FOREIGN KEY (username) REFERENCES user(username)  
    ON DELETE CASCADE ON UPDATE CASCADE  
);
```

```
CREATE TABLE IF NOT EXISTS languages(  
    candid VARCHAR(30) DEFAULT 'unknown' NOT NULL,  
    lang SET ('EN', 'FR', 'SP', 'GE', 'CH', 'GR'),  
    PRIMARY KEY(candid,lang),  
    CONSTRAINT candid_languages FOREIGN KEY (candid) REFERENCES employee(username)  
    ON DELETE CASCADE ON UPDATE CASCADE  
);
```

```
CREATE TABLE IF NOT EXISTS project(  
    candid VARCHAR(30) DEFAULT 'unknown' NOT NULL,  
    num TINYINT(4) DEFAULT '0' NOT NULL,  
    descr TEXT NOT NULL,  
    url VARCHAR(60) DEFAULT 'unknown' NOT NULL,  
    PRIMARY KEY(candid,num),  
    CONSTRAINT candid_project FOREIGN KEY (candid) REFERENCES employee(username)  
    ON DELETE CASCADE ON UPDATE CASCADE  
);
```

```
CREATE TABLE IF NOT EXISTS degree (  
    titlos VARCHAR(150) DEFAULT 'unknown' NOT NULL,  
    idryma VARCHAR(140) DEFAULT 'unknown' NOT NULL,  
    bathmida ENUM('BSc','MSc','PhD'),
```

```

PRIMARY KEY(titlos, idryma)
);

CREATE TABLE IF NOT EXISTS has_degree(
    degr_title VARCHAR(150) DEFAULT 'unknown' NOT NULL,
    degr_idryma VARCHAR(140) DEFAULT 'unknown' NOT NULL,
    cand_username VARCHAR(30) DEFAULT 'unknown' NOT NULL,
    etos YEAR(4) DEFAULT '0' NOT NULL,
    grade FLOAT DEFAULT '0' NOT NULL,
    PRIMARY KEY(degr_title, degr_idryma, cand_username),
    CONSTRAINT has_degree_username FOREIGN KEY (degr_title, degr_idryma) REFERENCES degree(titlos, idryma)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT has_degree1_username FOREIGN KEY (cand_username) REFERENCES employee(username)
        ON DELETE CASCADE ON UPDATE CASCADE
);

```

```

CREATE TABLE IF NOT EXISTS applies (
    cand_username VARCHAR(30) DEFAULT 'unknown' NOT NULL,
    job_id INT(11) DEFAULT '0' NOT NULL,
    PRIMARY KEY (cand_username, job_id),
    CONSTRAINT applies_username FOREIGN KEY (cand_username) REFERENCES employee(username)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT applies_job FOREIGN KEY (job_id) REFERENCES job(id)
        ON DELETE CASCADE ON UPDATE CASCADE
);

```

```

INSERT INTO etaireia (AFM, DOY, name, tel, street, num, city, country) VALUES
    ('265032148', 'Patrwn', 'Nikolaou', '2610486522', 'Mezonos', '59', 'Patras', 'Greece'),
    ('623946215', 'Patrwn', 'Papadopoylos kai sia', '2103625577', 'Ermou', '150', 'Athens', 'Greece'),
    ('203492781', 'Patrwn', 'Balafouths AE', '2610885423', 'Agiou Nikolaou', '23', 'Patras', 'Greece');

```

```

INSERT INTO user (username, password, name, lastname, reg_date, email) VALUES
    ('Douvris', 'Jdjoswkds', 'Fotis', 'Douvris', '2009-12-14', 'douvrisf@yahoo.com'),
    ('Georgiou', '15121996', 'Aggelikh', 'Georgiou', '2004-06-06', 'aggelikhgeorgiou@gmail.com'),
    ('Androutsos', 'Kddkdekl', 'Thodorhs', 'Androutsos', '2010-01-09', 'thodandrou@gmail.com'),

```

```

('Dhnhtriou', 'Panagiota29', 'Panagiota', 'Dhnhtriou', '2018-05-25',
'panagiotadhnhtriou@gmail.com'),
('Andrikopoulos', 'Lakksood', 'Dhnhtrhs', 'Andrikopoulos', '2022-02-22',
'megalosdhnhtrhs@yahoo.com'),
('Katsaros', '20202020', 'Alejandros', 'Katsaros', '2007-11-07', 'alejandroskats@gmail.com'),
('Xristopoulos', 'Xmlmjkd', 'Nikolas', 'Xristopoulos', '2022-01-01', 'nikolasxrist@gmail.com'),
('Lious', 'Fr@nk1i0us', 'Frank', 'Lious', '2023-02-01', 'liousfrank@gmail.com'),
('Papanikolaou', 'Nt1n@26', 'Konstantina', 'Papanikolaou', '2022-03-01',
'ntinapapa@yahoo@gmail.com'),
('Nikolakopoulou', 'Anna1985', 'Marianna', 'Nikolakopoulou', '2022-04-01',
'Marianna1985@yahoo.com'),
('Ioannidh', '56426348', 'Hlianna', 'Ioannidh', '2022-05-01', 'ioannidhhl@yahoo.com'),
('Papapaulou', 'Mariamaria', 'Maria', 'Papapaulou', '2022-06-01', 'Mariapapa@gmail.com');

```

INSERT INTO employee (username, bio, sistatikes, certificates) VALUES

```

('Xristopoulos', 'XristopoulosN.docx', 'sistatikhXRIST.docx', 'pistXristopoulos.docx'),
('Lious', 'Lious.docx', 'sistatikhLious.docx', 'LiousCertificates.docx'),
('Papanikolaou', 'Papanikolaou.docx', 'sistatikhPapanikolaou.docx', 'certPapanikolaou.docx'),
('Nikolakopoulou', 'Nikolakopoulou.pdf', 'sistatikhMarianna.pdf', 'DEFAULT'),
('Ioannidh', 'Ioannidh.pdf', 'DEFAULT', 'IoannidhCertificates.pdf'),
('Papapaulou', 'PapapaulouBio.pdf', 'PapapaulouSist.pdf', 'DEFAULT');

```

INSERT INTO evaluator (username, exp_years, firm) VALUES

```

('Douvris', '15', '265032148'),
('Georgiou', '20', '623946215'),
('Androutsos', '14', '203492781'),
('Dhnhtriou', '6', '265032148'),
('Andrikopoulos', '2', '203492781'),
('Katsaros', '17', '623946215');

```

INSERT INTO degree (titlos, idryma, bathmida) VALUES

```

('Computer Engineering and Informatics', 'Panepisthmio Patrwn', 'BSc'),
('Electrical Engineering', 'Panepisthmio Patrwn', 'MSc'),
('Mathematics', 'Panepisthmio Patrwn', 'BSc'),
('Logistics', 'Panepisthmio Patrwn', 'BSc'),
('Architecture', 'Panepisthmio Patrwn', 'PhD'),
('Economics', 'Panepisthmio Patrwn', 'MSc');

```

```

INSERT INTO job (start_date, salary, position, edra, evaluator, announce_date, submission_date) VALUES
('2022-01-01', '67000', 'Ypeuthinos C', 'Patra', 'Androutsos', '2022-01-01', '2024-12-31' ),
('2022-02-01', '40000', 'Ypeuthinos G', 'Patra', 'Georgiou', '2022-02-01', '2024-12-31' ),
('2022-03-01', '102000', 'Ypeuthinos A', 'Patra', 'Douvris', '2022-03-01', '2024-12-31' ),
('2022-04-01', '54000', 'Ypeuthinos F', 'Patra', 'Androutsos', '2022-04-01', '2024-12-31'),
('2022-05-01', '65000', 'Ypeuthinos D', 'Patra', 'Dhmhtriou', '2022-05-01', '2024-12-31' ),
('2022-06-01', '58000', 'Ypeuthinos E', 'Patra', 'Andrikopoulos', '2022-06-01', '2024-12-31' ),
('2022-07-01', '92000', 'Ypeuthinos B', 'Patra', 'Katsaros', '2022-07-01', '2024-12-31' ),
('2022-08-01', '58000', 'Ypeuthinos E', 'Patra', 'Katsaros', '2022-08-01', '2024-12-31' );

```

```

INSERT INTO has_degree (degr_title, degr_idryma, cand_username, etos, grade) VALUE
('Computer Engineering and Informatics', 'Panepisthmio Patrwn', 'Xristopoulos', '2023', '8'),
('Economics', 'Panepisthmio Patrwn', 'Lious', '2022', '9'),
('Architecture', 'Panepisthmio Patrwn', 'Nikolakopoulou', '2023', '8.6'),
('Electrical Engineering', 'Panepisthmio Patrwn', 'Papapaulou', '2023', '6');

```

```

INSERT INTO languages (candid, lang) VALUES
('Xristopoulos', 'EN'),
('Lious', 'GR'),
('Nikolakopoulou', 'SP'),
('Papapaulou', 'CH');

```

```

INSERT INTO project (candid, num, descr, url) VALUES
('Xristopoulos', '1', 'xristopoulos.pdf', 'https://XRproject.com' ),
('Lious', '2', 'lious1.pdf', 'https://LI1project.com' ),
('Lious', '3', 'lious2.pdf', 'https://LI2project.com' ),
('Papanikolaou', '4', 'papanikolaou.pdf', 'https://PAPAPproject.com' ),
('Nikolakopoulou', '5', 'nikolakopoulou.pdf', 'https://NIKOLproject.com' ),
('Ioannidh', '6', 'ioannidh.pdf', 'https://IOANproject.com' ),
('Papapaulou', '7', 'papapaulou.pdf', 'https://PAPAPAPproject.com' );

```

```

INSERT INTO subject (title, descr, belongs_to) VALUES
('Web Developer', 'This job requires experience in designing, developing, and maintaining web applications.', null),
('Product Manager', 'This job requires experience in defining, prioritizing, and launching new products.', null),

```


('Marketing Specialist', 'This job requires experience in creating and executing marketing campaigns to promote products and services.', null),

('Customer Support Specialist', 'This job requires experience in providing technical and customer support to resolve issues and answer questions.', null),

('Accountant', 'This job requires experience in managing financial records, preparing reports, and ensuring financial compliance.', null),

('Human Resources Manager', 'This job requires experience in recruiting, onboarding, and managing employees.', null);

INSERT INTO requires (job_id , subject_title) VALUES

('1','Web Development'),

('2','Marketing Specialist'),

('3','Human Resources Manager'),

('4','Web Development'),

('5','Accountant'),

('6','Customer Support Specialist'),

('7','Product Manager'),

('8','Web Development');

INSERT INTO applies (cand_username, job_id) VALUES

('Xristopoulos', '3'),

('Lious', '1'),

('Papanikolaou', '3'),

('Nikolakopoulou', '5'),

('Ioannidh', '3'),

('Papapoulou', '8');

-- 3.1.2 ΝΕΕΣ ΑΠΑΙΤΗΣΕΙΣ

-- ΕΡΩΤΗΜΑ 3.1.2.1 --

-- Drop foreign key constraints

ALTER TABLE applies

DROP FOREIGN KEY applies_username,

DROP FOREIGN KEY applies_job;

-- Drop the existing primary key

ALTER TABLE applies

DROP PRIMARY KEY;

-- Add a new column 'id' with AUTO_INCREMENT and make it the new primary key

ALTER TABLE applies

ADD COLUMN id INT(11) AUTO_INCREMENT PRIMARY KEY FIRST;

-- Add new columns 'status' and 'application_date'

ALTER TABLE applies

ADD COLUMN status ENUM('active', 'completed', 'cancelled') DEFAULT 'active' NOT NULL AFTER job_id,

ADD COLUMN application_date DATETIME DEFAULT CURRENT_TIMESTAMP AFTER status;

-- Rename the table to 'application_details'

RENAME TABLE applies TO application_details;

/* The new applies table

CREATE TABLE IF NOT EXISTS application_details (

id INT(11) AUTO_INCREMENT NOT NULL,

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

job_id INT(11) DEFAULT '0' NOT NULL,

status ENUM('active', 'completed', 'cancelled') DEFAULT 'active' NOT NULL,

application_date DATETIME DEFAULT CURRENT_TIMESTAMP,

PRIMARY KEY (id),

CONSTRAINT app_details_username FOREIGN KEY (cand_username) REFERENCES employee(username)

ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT app_details_job FOREIGN KEY (job_id) REFERENCES job(id)

ON DELETE CASCADE ON UPDATE CASCADE

);

*/

DELIMITER //

CREATE TRIGGER before_insert_application

BEFORE INSERT ON application_details

FOR EACH ROW

BEGIN

```

DECLARE active_applications_count INT;
DECLARE job_start_date DATETIME;

-- Count the number of active applications for the employee
SELECT COUNT(*)
INTO active_applications_count
FROM application_details
WHERE cand_username = NEW.cand_username
    AND status = 'active';

-- Check if the employee already has three active applications
IF active_applications_count >= 3 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'An employee cannot have more than three active applications.';
END IF;

-- Check if the application date is less than 15 days before the job start date
SELECT start_date
INTO job_start_date
FROM job
WHERE id = NEW.job_id;

IF NEW.application_date > DATE_SUB(job_start_date, INTERVAL 15 DAY) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'New applications must be submitted at least 15 days before the job start date.';
END IF;

END//

DELIMITER ;

DELIMITER //
CREATE TRIGGER before_update_application
BEFORE UPDATE ON application_details
FOR EACH ROW
BEGIN

```

```

DECLARE active_applications_count INT;
DECLARE job_start_date DATETIME;

-- Count the number of active applications for the employee
SELECT COUNT(*)
INTO active_applications_count
FROM application_details
WHERE cand_username = NEW.cand_username
    AND status = 'active';

-- Check if the employee already has three active applications
IF active_applications_count >= 3 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'An employee cannot have more than three active applications.';
END IF;

-- Check if the application is being canceled
IF NEW.status = 'cancelled' THEN
    -- Check if the cancellation date is less than 10 days before the job start date
    SELECT start_date
    INTO job_start_date
    FROM job
    WHERE id = NEW.job_id;

    IF NEW.application_date > DATE_SUB(job_start_date, INTERVAL 10 DAY) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'An application cannot be canceled within 10 days before the job start date.';
    END IF;
END IF;

-- Check if the canceled application is being reactivated and the user has less than 3 active applications
IF NEW.status = 'active' AND active_applications_count >= 3 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'An employee cannot have more than three active applications.';
END IF;

END//

```

DELIMITER ;

-- EPQTHMA 3.1.2.2

```
CREATE TABLE IF NOT EXISTS evaluation_assignments (  
    id INT(11) AUTO_INCREMENT NOT NULL,  
    job_id INT(11) NOT NULL,  
    evaluator1_username VARCHAR(30) NOT NULL,  
    evaluator2_username VARCHAR(30) NOT NULL,  
    PRIMARY KEY (id),  
    CONSTRAINT eval_assign_job FOREIGN KEY (job_id) REFERENCES job(id) ON DELETE CASCADE ON  
UPDATE CASCADE  
);
```

-- Inserting evaluators for a specific job ID

```
INSERT INTO evaluation_assignments (job_id, evaluator1_username, evaluator2_username)  
VALUES  
    (1, 'Andrikopoulos', 'Androutsos'),  
    (2, 'Dhnhtriu', 'Douvris'),  
    (3, 'Georgiou', 'Katsaros'),  
    (4, 'Douvris', 'Georgiou'),  
    (5, 'Katsaros', 'Androutsos'),  
    (6, 'Dhnhtriu', 'Andrikopoulos');
```

```
CREATE TABLE IF NOT EXISTS historical_applications (  
    id INT(11) AUTO_INCREMENT NOT NULL,  
    evaluator1_username VARCHAR(30) DEFAULT 'unknown' NOT NULL,  
    evaluator2_username VARCHAR(30) DEFAULT 'unknown' NOT NULL,  
    cand_username VARCHAR(30) DEFAULT 'unknown' NOT NULL,  
    job_id INT(11) DEFAULT '0' NOT NULL,  
    status ENUM('active', 'completed', 'canceled') DEFAULT 'completed' NOT NULL,  
    application_date DATETIME DEFAULT CURRENT_TIMESTAMP,  
    final_rating INT NOT NULL DEFAULT 0,  
    PRIMARY KEY (id),  
    CONSTRAINT hist_app_username FOREIGN KEY (cand_username) REFERENCES employee(username)  
ON DELETE CASCADE ON UPDATE CASCADE,
```

```
CONSTRAINT hist_app_job FOREIGN KEY (job_id) REFERENCES job(id)
ON DELETE CASCADE ON UPDATE CASCADE
);
```

```
DELIMITER //
```

```
CREATE PROCEDURE check_application_status(IN candidate_username VARCHAR(30), IN job_id INT,
OUT app_status VARCHAR(20))
```

```
BEGIN
```

```
-- Get the status of the application
```

```
SELECT COALESCE(status, 'not_found')
```

```
INTO app_status
```

```
FROM application_details
```

```
WHERE cand_usrname = candidate_username AND job_id = job_id
```

```
LIMIT 1;
```

```
END //
```

```
DELIMITER ;
```

```
DELIMITER //
```

```
CREATE PROCEDURE evaluate_promotion(
```

```
IN candidate_username VARCHAR(30),
```

```
IN job_id INT
```

```
)
```

```
BEGIN
```

```
DECLARE evaluator1_username VARCHAR(30);
```

```
DECLARE evaluator2_username VARCHAR(30);
```

```
DECLARE rating1 INT;
```

```
DECLARE rating2 INT;
```

```
DECLARE final_rating INT;
```

```
DECLARE app_status VARCHAR(20);
```

```
DECLARE job_exists INT;
```

```
-- Check if the job_id exists in the applications
```

```
SELECT COUNT(*)
```

```
INTO job_exists
```

```

FROM application_details
WHERE job_id = job_id;

-- If the job_id does not exist, print an error message and exit the procedure
IF job_exists = 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Job does not exist in applications. Skipping evaluation.';
END IF;

-- Check if evaluators are assigned for the given job_id
SELECT ea.evaluator1_username, ea.evaluator2_username
INTO evaluator1_username, evaluator2_username
FROM evaluation_assignments ea
WHERE ea.job_id = job_id
LIMIT 1;

-- If evaluators are not assigned, print an error message and exit the procedure
IF evaluator1_username IS NULL OR evaluator2_username IS NULL THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Evaluators are not assigned for the given job_id. Skipping evaluation.';
END IF;

-- Call the procedure to check the application status
CALL check_application_status(candidate_username, job_id, app_status);

-- Check if the application is cancelled or completed
IF TRIM(app_status) = 'cancelled' THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Application is cancelled. Skipping evaluation.';
ELSEIF TRIM(app_status) = 'completed' THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Application is already completed. Skipping evaluation.';
ELSE
    -- Generate random ratings between 1 and 20 for both evaluators, with a possibility of NULL
    SET rating1 = (CASE WHEN RAND() < 0.8 THEN FLOOR(1 + RAND() * 20) ELSE NULL END);
    SET rating2 = (CASE WHEN RAND() < 0.8 THEN FLOOR(1 + RAND() * 20) ELSE NULL END);

```

```

-- Calculate the final rating based on qualifications
SELECT (SUM(IF(dg.bathmida = 'BSc', 1, IF(dg.bathmida = 'MSc', 2, IF(dg.bathmida = 'PhD', 3, 0)))) +
        (SELECT COUNT(*) FROM languages l WHERE l.candid = candidate_username) +
        (SELECT COUNT(*) FROM project p WHERE p.candid = candidate_username))
INTO final_rating
FROM has_degree hd
JOIN degree dg ON hd.degr_title = dg.titlos AND hd.degr_idryma = dg.idryma
WHERE hd.cand_usrname = candidate_username;

-- If one rating is missing, fill in based on qualifications
IF rating1 IS NULL AND rating2 IS NOT NULL THEN
    SET rating1 = ((SUM(IF(dg.bathmida = 'BSc', 1, IF(dg.bathmida = 'MSc', 2, IF(dg.bathmida = 'PhD', 3,
0)))) +
        (SELECT COUNT(*) FROM languages l WHERE l.candid = candidate_username) +
        (SELECT COUNT(*) FROM project p WHERE p.candid = candidate_username)));
END IF;
IF rating2 IS NULL AND rating1 IS NOT NULL THEN
    SET rating2 = ((SUM(IF(dg.bathmida = 'BSc', 1, IF(dg.bathmida = 'MSc', 2, IF(dg.bathmida = 'PhD', 3,
0)))) +
        (SELECT COUNT(*) FROM languages l WHERE l.candid = candidate_username) +
        (SELECT COUNT(*) FROM project p WHERE p.candid = candidate_username)));
END IF;
IF rating1 IS NULL AND rating2 IS NULL THEN
    -- Calculate the default rating based on qualifications
    SET final_rating = ((SUM(IF(dg.bathmida = 'BSc', 1, IF(dg.bathmida = 'MSc', 2, IF(dg.bathmida =
'PhD', 3, 0)))) +
        (SELECT COUNT(*) FROM languages l WHERE l.candid = candidate_username) +
        (SELECT COUNT(*) FROM project p WHERE p.candid = candidate_username)));
ELSE
    -- Use the provided ratings if available
    SET final_rating = ((COALESCE(rating1, 0) + COALESCE(rating2, 0)) / 2);
END IF;

-- Set session variables for the ratings and final rating
SET @rating1 = rating1;
SET @rating2 = rating2;
SET @final_rating = final_rating;

```



```

-- Print the ratings and final rating, including the candidate's username
SELECT candidate_username AS 'Candidate Username', @rating1 AS 'Evaluator 1 Rating', @rating2
AS 'Evaluator 2 Rating', @final_rating AS 'Final Rating';

END IF;

END //

DELIMITER;

DELIMITER //

CREATE PROCEDURE move_completed_applications()
BEGIN
-- Move completed applications to historical_applications table
INSERT INTO historical_applications (cand_username, job_id, status, application_date)
SELECT cand_username, job_id, status, application_date
FROM application_details
WHERE status = 'completed';

-- Get IDs of completed applications
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SET @completed_ids = (SELECT GROUP_CONCAT(id) FROM application_details WHERE status =
'completed');
SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;

-- Delete completed applications from application_details table
IF @completed_ids IS NOT NULL THEN
SET @delete_query = CONCAT('DELETE FROM application_details WHERE id IN (', @completed_ids,
')');

-- Execute the delete query
PREPARE stmt FROM @delete_query;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
END IF;
END //

DELIMITER ;

```

-- Call the procedure to move completed applications

CALL move_completed_applications();

INSERT INTO application_details (cand_username, job_id, status, application_date)

VALUES

('Lious', 1, 'active', '2021-01-01'),

('Ioannidh', 2, 'cancelled', '2021-01-02'),

('Papapaulou', 4, 'active', '2021-01-03');

/*

CALL evaluate_promotion('Lious', 1);

CALL evaluate_promotion('Ioannidh', 3);

CALL evaluate_promotion('Papapaulou', 5);

*/

-- EPΩTHMA 3.1.2.3

-- The following table has already been created but will also be placed here just for reference

CREATE TABLE IF NOT EXISTS historical_applications (

id INT(11) AUTO_INCREMENT NOT NULL,

evaluator1_username VARCHAR(30) DEFAULT 'unknown' NOT NULL,

evaluator2_username VARCHAR(30) DEFAULT 'unknown' NOT NULL,

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

job_id INT(11) DEFAULT '0' NOT NULL,

status ENUM('active', 'completed', 'canceled') DEFAULT 'completed' NOT NULL,

application_date DATETIME DEFAULT CURRENT_TIMESTAMP,

final_rating INT NOT NULL DEFAULT 0,

PRIMARY KEY (id),

CONSTRAINT hist_app_username FOREIGN KEY (cand_username) REFERENCES employee(username)

ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT hist_app_job FOREIGN KEY (job_id) REFERENCES job(id)

ON DELETE CASCADE ON UPDATE CASCADE

);

DELIMITER //

DELIMITER //

```

CREATE PROCEDURE generate_historical_applications()
BEGIN
    DECLARE i INT DEFAULT 0;
    DECLARE batch_size INT DEFAULT 500;

    WHILE i <= 60000 DO
        START TRANSACTION;

        -- Insert into historical_applications with both evaluators
        INSERT INTO historical_applications (evaluator1_username, evaluator2_username, cand_username,
        job_id, status, final_rating)
        SELECT
            (SELECT evaluator1_username FROM evaluation_assignments ORDER BY RAND() LIMIT 1) AS
            evaluator1_username,
            (SELECT evaluator2_username FROM evaluation_assignments WHERE evaluator2_username <>
            evaluator1_username ORDER BY RAND() LIMIT 1) AS evaluator2_username,
            (SELECT username FROM employee ORDER BY RAND() LIMIT 1) AS cand_username,
            (SELECT id FROM job ORDER BY RAND() LIMIT 1) AS job_id,
            'completed' AS status,
            FLOOR(1 + RAND() * 20) AS final_rating
        FROM information_schema.tables ORDER BY RAND() LIMIT batch_size;

        COMMIT;

        SET i = i + batch_size;
    END WHILE;
END //

DELIMITER ;

-- Call the procedure to generate historical applications
/* CALL generate_historical_applications();
SELECT * FROM historical_applications;
*/
-- EPΩTHMA 3.1.2.4

```

```
CREATE TABLE IF NOT EXISTS dba (  
    id INT(11) AUTO_INCREMENT NOT NULL,  
    username VARCHAR(30) NOT NULL,  
    start_date DATE NOT NULL,  
    end_date DATE,  
    PRIMARY KEY (id),  
    UNIQUE KEY (username)  
);
```

```
INSERT INTO dba (username, start_date) VALUES ('theokatsa', '2024-01-10');  
INSERT INTO dba (username, start_date) VALUES ('aggidouври', '2024-01-10');
```

-- 3.1.3 Stored Procedures

-- 3.1.3.1

DELIMITER //

```
CREATE PROCEDURE CheckEvaluation(  
    IN p_evaluator_username VARCHAR(255),  
    IN p_employee_username VARCHAR(255),  
    IN p_job_id INT,  
    OUT p_result INT  
)  
BEGIN  
    DECLARE message_text VARCHAR(255); -- Declare a variable to hold the message  
  
    -- Check if the evaluator exists for the specified job_id in evaluation_assignments  
    SELECT COUNT(*)  
    INTO p_result  
    FROM evaluation_assignments  
    WHERE evaluator1_username = p_evaluator_username OR evaluator2_username = p_evaluator_username  
    AND job_id = p_job_id;  
  
    -- If the evaluator does not exist for the specified job_id, set the result to 0
```

```

IF p_result = 0 THEN
    SET p_result = 0;
    SIGNAL SQLSTATE '45000'
    SET message_text = 'Evaluator not assigned to the specified job.';
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = message_text;

ELSE
    -- Check if the employee exists
    SELECT COUNT(*)
    INTO p_result
    FROM employee
    WHERE username = p_employee_username;

    -- If the employee does not exist, raise an error
    IF p_result = 0 THEN
        SIGNAL SQLSTATE '45000'
        SET message_text = 'Invalid employee username.';
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = message_text;

    ELSE
        -- Check if the evaluation exists for the specified evaluator, employee, and job
        SELECT final_rating
        INTO p_result
        FROM historical_applications
        WHERE (evaluator1_username = p_evaluator_username OR evaluator2_username =
p_evaluator_username)
        AND cand_usrname = p_employee_username
        AND job_id = p_job_id
        LIMIT 1;

        -- If an evaluation exists, print a message and return the rating
        IF p_result IS NOT NULL THEN
            SET message_text = CONCAT('Evaluation already exists. Rating: ', p_result);
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = message_text;

        ELSE
            -- Calculate the rating using evaluate_promotion procedure
            CALL evaluate_promotion(p_evaluator_username, p_employee_username, p_job_id, p_result);

```

```

-- Print a message indicating that the evaluation was calculated
SELECT 'Evaluation calculated using evaluate_promotion' AS message;

    END IF;
END IF;
END IF;
END //

DELIMITER ;

-- EPΩTHMA 3.1.3.2

DELIMITER //

CREATE PROCEDURE ManageApplication(
    IN p_employee_username VARCHAR(30),
    IN p_job_id INT,
    IN p_action CHAR(1)
)
BEGIN
    DECLARE v_evaluator1_username VARCHAR(30);
    DECLARE v_evaluator2_username VARCHAR(30);
    DECLARE v_application_status VARCHAR(20);

    -- Check if the action is valid ('i', 'c', or 'a')
    IF NOT (p_action IN ('i', 'c', 'a')) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Invalid action. Use "i" for insert, "c" for cancel, or "a" for activate.';
    END IF;

    -- Check if the employee exists
    IF NOT EXISTS (SELECT 1 FROM employee WHERE username = p_employee_username) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Invalid employee username.';
    END IF;

    -- Check if the job exists
    IF NOT EXISTS (SELECT 1 FROM job WHERE id = p_job_id) THEN

```

```

    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Invalid job ID.';
END IF;

-- Get the evaluators for the job
SELECT evaluator1_username, evaluator2_username
INTO v_evaluator1_username, v_evaluator2_username
FROM evaluation_assignments
WHERE job_id = p_job_id
LIMIT 1;

-- If evaluators are not assigned, fill them in from the same company
IF v_evaluator1_username IS NULL THEN
    SELECT username
    INTO v_evaluator1_username
    FROM evaluator
    WHERE firm = (SELECT firm FROM evaluator WHERE username = p_employee_username)
    AND username <> p_employee_username
    LIMIT 1;
END IF;

IF v_evaluator2_username IS NULL THEN
    SELECT username
    INTO v_evaluator2_username
    FROM evaluator
    WHERE firm = (SELECT firm FROM evaluator WHERE username = p_employee_username)
    AND username <> p_employee_username
    AND username <> v_evaluator1_username
    LIMIT 1;
END IF;

-- Call the procedure to check the application status
CALL check_application_status(p_employee_username, p_job_id, v_application_status);

-- Perform actions based on the specified action
CASE v_application_status
    WHEN 'not_found' THEN

```

```

-- Insert an application
IF p_action = 'i' THEN
    -- Start a transaction if necessary
    INSERT INTO application_details (cand_username, job_id, status, application_date)
    VALUES (p_employee_username, p_job_id, 'active', CURRENT_TIMESTAMP);
    -- Commit the transaction if necessary
    COMMIT;
END IF;
WHEN 'active' THEN
    -- Cancel an application
    IF p_action = 'c' THEN
        UPDATE application_details
        SET status = 'cancelled'
        WHERE cand_username = p_employee_username AND job_id = p_job_id;
        COMMIT;
    END IF;
WHEN 'cancelled' THEN
    -- Activate a canceled application
    IF p_action = 'a' THEN
        UPDATE application_details
        SET status = 'active'
        WHERE cand_username = p_employee_username AND job_id = p_job_id;
        COMMIT;
    END IF;
END CASE;
END //

```

```

DELIMITER ;

```

```

-- 3.1.3.3

```

```

CREATE TABLE IF NOT EXISTS employee_jobs (
    id INT(11) AUTO_INCREMENT NOT NULL,
    job_id INT(11) NOT NULL,
    employee_username VARCHAR(30) NOT NULL,
    rating DECIMAL(5, 2) NOT NULL,

```



```

selection_date DATETIME DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY (id),
CONSTRAINT emp_jobs_job FOREIGN KEY (job_id) REFERENCES job(id) ON DELETE CASCADE ON
UPDATE CASCADE,
CONSTRAINT emp_jobs_employee FOREIGN KEY (employee_username) REFERENCES
employee(username) ON DELETE CASCADE ON UPDATE CASCADE
);

```

```

CREATE TABLE IF NOT EXISTS job_positions (
    job_id INT(11) AUTO_INCREMENT NOT NULL,
    job_title VARCHAR(255) NOT NULL,
    status VARCHAR(20) DEFAULT 'open',
    PRIMARY KEY (job_id)
);

```

```

DELIMITER //

```

```

CREATE PROCEDURE fill_job_position(
    IN p_job_id INT
)
BEGIN
    DECLARE candidate_username VARCHAR(30);
    DECLARE app_status VARCHAR(20);
    DECLARE evaluator1_username VARCHAR(30);
    DECLARE evaluator2_username VARCHAR(30);
    DECLARE evaluation_result INT;

    -- Check if the job_id exists in the applications
    DECLARE job_exists INT;
    SELECT COUNT(*) INTO job_exists
    FROM application_details
    WHERE job_id = p_job_id;

    -- If the job_id does not exist, print an error message and exit the procedure
    IF job_exists = 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Job does not exist in applications. Cannot fill position.';
    END IF;
END
//

```

```

ELSE
    -- Check if the job is already occupied
    SELECT COUNT(*) INTO @job_occupied
    FROM employee_jobs
    WHERE job_id = p_job_id;

    IF @job_occupied > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Job is already occupied. Cannot fill position.';
    ELSE
        -- Get the candidate's username from historical_applications
        SELECT cand_username INTO candidate_username
        FROM historical_applications
        WHERE job_id = p_job_id
        ORDER BY final_rating DESC
        LIMIT 1;

        -- Call the procedure to check the application status
        CALL check_application_status(candidate_username, p_job_id, app_status);

        -- Check if the application is cancelled or completed
        IF TRIM(app_status) = 'cancelled' THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Selected candidate has cancelled application. Cannot fill position.';
        ELSEIF TRIM(app_status) = 'completed' THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Selected candidate has already completed the application. Cannot fill
position.';
        ELSE
            -- Check if evaluators are assigned for the given job_id
            SELECT ea.evaluator1_username, ea.evaluator2_username
            INTO evaluator1_username, evaluator2_username
            FROM evaluation_assignments ea
            WHERE ea.job_id = p_job_id
            LIMIT 1;

            -- If evaluators are not assigned, print an error message and exit the procedure

```

```

IF evaluator1_username IS NULL OR evaluator2_username IS NULL THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Evaluators are not assigned for the given job_id. Cannot fill position.';
END IF;

-- Check if the evaluation exists for the specified evaluator, employee, and job
SELECT COALESCE(final_rating, 0)
INTO evaluation_result
FROM historical_applications
WHERE (evaluator1_username = evaluator1_username OR evaluator2_username =
evaluator2_username)
AND cand_username = candidate_username
AND job_id = p_job_id
LIMIT 1;

-- If the evaluation does not exist, print an error message and exit the procedure
IF evaluation_result = 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Evaluation does not exist. Cannot fill position.';
ELSE
    -- Call the procedure to evaluate promotion
    CALL evaluate_promotion(candidate_username, p_job_id);

    -- Update the job status as filled
    UPDATE job_positions
    SET status = 'filled'
    WHERE job_id = p_job_id;

    -- Insert the job assignment into the employee_jobs table
    INSERT INTO employee_jobs (job_id, employee_username, rating)
    SELECT p_job_id, cand_username, final_rating
    FROM historical_applications
    WHERE job_id = p_job_id
    ORDER BY final_rating DESC
    LIMIT 1;

    -- Get the ratings for the selected candidate

```

```

SELECT final_rating, final_rating AS 'Evaluator 1 Rating', final_rating AS 'Evaluator 2 Rating'
INTO @final_rating, @rating1, @rating2
FROM historical_applications
WHERE job_id = p_job_id
ORDER BY final_rating DESC
LIMIT 1;

```

-- Print a success message with ratings and final evaluation

```

SELECT CONCAT(
    'Job position filled for job_id: ', p_job_id,
    ' with candidate: ', candidate_username,
    '. Evaluator 1 Rating: ', @rating1,
    '. Evaluator 2 Rating: ', @rating2,
    '. Final Rating: ', @final_rating
) AS 'Success Message';

```

END IF;

END IF;

END IF;

END IF;

END //

DELIMITER ;

-- EPΩTHMA 3.1.3.4

-- Create the index on final_rating column

```
CREATE INDEX idx_final_rating ON historical_applications (final_rating);
```

```
CREATE INDEX idx_evaluator_filter ON historical_applications (evaluator1_username,
evaluator2_username);
```

-- a)

DELIMITER //

```
CREATE PROCEDURE filter_applications_by_ratings(IN min_rating INT, IN max_rating INT)
```

```
BEGIN
```

-- Select cand_usurname, job_id, and final_rating based on the rating range

```
SELECT cand_usurname AS employee_username, job_id, final_rating
```

```
FROM historical_applications USE INDEX (idx_final_rating)
```

```
WHERE final_rating BETWEEN min_rating AND max_rating
ORDER BY final_rating; -- Optional: Add ORDER BY if needed
END //
```

```
DELIMITER ;
```

```
-- b)
DELIMITER //
```

```
CREATE PROCEDURE filter_applications_by_evaluator(IN evaluator_username VARCHAR(30))
BEGIN
    -- Select cand_usurname (employee username) and job_id for applications evaluated by the specified evaluator
    SELECT cand_usurname AS employee_username, job_id
    FROM historical_applications USE INDEX (idx_evaluator_filter)
    WHERE evaluator1_username = evaluator_username OR evaluator2_username = evaluator_username;
END //
```

```
DELIMITER ;
```

```
-- EPΩTHMA 3.1.4.
```

```
-- 3.1.4.1
```

```
SELECT * FROM dba;
SET @current_admin = 'theokatsa';
SET @current_admin_id = '1';
-- SET @current_admin = 'douvriaggi';
-- SET @current_admin_id = '2';
```

```
CREATE TABLE IF NOT EXISTS dba_log (
    log_id INT(11) AUTO_INCREMENT NOT NULL,
    dba_id INT(11) NOT NULL,
    username VARCHAR(30) NOT NULL,
    table_name VARCHAR(30) NOT NULL,
    action_description VARCHAR(255) NOT NULL,
    action_date DATETIME NOT NULL,
```

```
PRIMARY KEY (log_id),  
FOREIGN KEY (dba_id) REFERENCES dba(id) ON DELETE CASCADE ON UPDATE CASCADE  
);
```

```
DELIMITER //
```

```
CREATE TRIGGER job_after_insert  
AFTER INSERT ON job  
FOR EACH ROW  
BEGIN  
    INSERT INTO dba_log (dba_id, username, action_description, table_name, action_date)  
    VALUES (@current_admin_id, @current_admin, 'insert', 'job', NOW());  
END;  
//
```

```
CREATE TRIGGER job_after_update  
AFTER UPDATE ON job  
FOR EACH ROW  
BEGIN  
    INSERT INTO dba_log (dba_id, username, action_description, table_name, action_date)  
    VALUES (@current_admin_id, @current_admin, 'update', 'job', NOW());  
END;  
//
```

```
CREATE TRIGGER job_after_delete  
AFTER DELETE ON job  
FOR EACH ROW  
BEGIN  
    INSERT INTO dba_log (dba_id, username, action_description, table_name, action_date)  
    VALUES (@current_admin_id, @current_admin, 'delete', 'job', NOW());  
END;  
//
```

```
DELIMITER ;
```

```
DELIMITER //
```

```
CREATE TRIGGER user_after_insert
AFTER INSERT ON user
FOR EACH ROW
BEGIN
    INSERT INTO dba_log (dba_id, username, action_description, table_name, action_date)
    VALUES (@current_admin_id, @current_admin, 'insert', 'user', NOW());
END;
//
```

```
CREATE TRIGGER user_after_update
AFTER UPDATE ON user
FOR EACH ROW
BEGIN
    INSERT INTO dba_log (dba_id, username, action_description, table_name, action_date)
    VALUES (@current_admin_id, @current_admin, 'update', 'user', NOW());
END;
//
```

```
CREATE TRIGGER user_after_delete
AFTER DELETE ON user
FOR EACH ROW
BEGIN
    INSERT INTO dba_log (dba_id, username, action_description, table_name, action_date)
    VALUES (@current_admin_id, @current_admin, 'delete', 'user', NOW());
END;
//
```

```
DELIMITER ;
```

```
DELIMITER //
```

```
CREATE TRIGGER degree_after_insert
AFTER INSERT ON degree
FOR EACH ROW
BEGIN
    INSERT INTO dba_log (dba_id, username, action_description, table_name, action_date)
    VALUES (@current_admin_id, @current_admin, 'insert', 'degree', NOW());
```

```

END;
//

CREATE TRIGGER degree_after_update
AFTER UPDATE ON degree
FOR EACH ROW
BEGIN
    INSERT INTO dba_log (dba_id, username, action_description, table_name, action_date)
    VALUES (@current_admin_id, @current_admin, 'update', 'degree', NOW());
END;
//

CREATE TRIGGER degree_after_delete
AFTER DELETE ON degree
FOR EACH ROW
BEGIN
    INSERT INTO dba_log (dba_id, username, action_description, table_name, action_date)
    VALUES (@current_admin_id, @current_admin, 'delete', 'degree', NOW());
END;
//

DELIMITER ;

/* Triggers already exist
-- EPQTHMA 3.1.4.2

DELIMITER //
CREATE TRIGGER before_insert_application
BEFORE INSERT ON application_details
FOR EACH ROW
BEGIN
    DECLARE active_applications_count INT;
    DECLARE job_start_date DATETIME;

    -- Count the number of active applications for the employee
    SELECT COUNT(*)

```



```

    INTO active_applications_count
    FROM application_details
    WHERE cand_username = NEW.cand_username
       AND status = 'active';

-- Check if the employee already has three active applications
IF active_applications_count >= 3 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'An employee cannot have more than three active applications.';
END IF;

-- Check if the application date is less than 15 days before the job start date
SELECT start_date
    INTO job_start_date
    FROM job
    WHERE id = NEW.job_id;

IF NEW.application_date > DATE_SUB(job_start_date, INTERVAL 15 DAY) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'New applications must be submitted at least 15 days before the job start
date.';
END IF;

END//

DELIMITER ;

-- EPΩTHMA 3.1.4.3

DELIMITER //
CREATE TRIGGER before_update_application
BEFORE UPDATE ON application_details
FOR EACH ROW
BEGIN
    DECLARE active_applications_count INT;
    DECLARE job_start_date DATETIME;

```

```

-- Count the number of active applications for the employee
SELECT COUNT(*)
INTO active_applications_count
FROM application_details
WHERE cand_username = NEW.cand_username
    AND status = 'active';

-- Check if the employee already has three active applications
IF active_applications_count >= 3 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'An employee cannot have more than three active applications.';
END IF;

-- Check if the application is being canceled
IF NEW.status = 'cancelled' THEN
    -- Check if the cancellation date is less than 10 days before the job start date
    SELECT start_date
    INTO job_start_date
    FROM job
    WHERE id = NEW.job_id;

    IF NEW.application_date > DATE_SUB(job_start_date, INTERVAL 10 DAY) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'An application cannot be canceled within 10 days before the job start date.';
    END IF;
END IF;

-- Check if the canceled application is being reactivated and the user has less than 3 active applications
IF NEW.status = 'active' AND active_applications_count >= 3 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'An employee cannot have more than three active applications.';
END IF;

END//

DELIMITER

```

* /