Εργασία στις Βάσεις Δεδομένων

Εφαρμογή τοπικού πρωταθλήματος ποδοσφαίρου

ΚΑΓΙΑΦΑ ΑΝΝΑ

Προπτυχιακή φοιτήτρια τμήματος ΗΜΤΥ, Πανεπιστήμιο Πατρών, up1092761@ac.upatras.gr

ΜΕΝΔΡΙΝΟΣ ΑΝΔΡΕΑΣ

Προπτυχιακή φοιτητής τμήματος ΗΜΤΥ, Πανεπιστήμιο Πατρών, up1092653@ac.upatras.gr

Στην παρούσα εργασία περιγράφεται η ανάπτυξη μιας εφαρμογής τοπικού πρωταθλήματος ποδοσφαίρου, με βασικό πυρήνα τη σχεδίαση και υλοποίηση της σχετικής βάσης δεδομένων. Για την εν λόγω μοντελοποίηση αναπτύχθηκε το σχετικό μοντέλο οντοτήτων – συσχετίσεων, καθώς και το σχεσιακό σχήμα και στη συνέχεια η βάση μας υλοποιήθηκε σε SQLite.Τέλος, δημιουργήσαμε μέσω python μία διεπαφή που αποτελεί το γραφικό περιβάλλον της εφαρμογής μας. Στόχος της εφαρμογής καθίσταται η καταχώρηση δεδομένων για τα σωματεία, τους παίκτες, τα γήπεδα, καθώς και η αποτελεσματική οργάνωση των αγωνιστικών και των αγώνων, η καταγραφή των αποτελεσμάτων και η εξαγωγή χρήσιμων και αξιοσημείωτων στατιστικών για τους παίκτες, τις ομάδες και το πρωτάθλημα γενικότερα.

1. ΕΙΣΑΓΩΓΗ – ΠΕΡΙΓΡΑΦΗ ΜΙΚΡΟΚΟΣΜΟΥ

Για να σχεδιάσουμε τη βάση δεδομένων της εφαρμογής μας, έπρεπε σε πρώτη φάση να θέσουμε τα όρια του προβλήματός μας. Στην προσπάθειά μας, αυτή, θέλαμε ο μικρόκοσμος να αποτυπωθεί με τέτοιον τρόπο, ώστε να είναι αληθοφανής και να ανταποκρίνεται στα δεδομένα ενός πραγματικού τοπικού πρωταθλήματος ποδοσφαίρου μιας ΕΠΣ, το οποίο καθιστά το αντικείμενο της εργασίας ενδιαφέρον, αλλά και αρκετά απαιτητικό.

Στα πλαίσια, αυτά, αποφασίσαμε πως θα υπάρχει διαφορετικό πρωτάθλημα κάθε αγωνιστική περίοδο. Κάθε σεζόν, λοιπόν, θα απαρτίζεται από αγωνιστικές, σε καθεμία από τις οποίες θα διεξάγονται αγώνες μεταξύ των ομάδων που συμμετέχουν στο πρωτάθλημα και αγωνίζονται στην ίδια κατηγορία. Πιο συγκεκριμένα, διακρίνουμε τρεις κατηγορίες, Α’, Β’ και Γ’, καθεμία από τις οποίες αποτελείται από 12 σωματεία. Αναφορικά με τα σωματεία, η ομάδα που κατεβάζει σε κάθε σεζόν στο πρωτάθλημα διαφοροποείται, καθώς μπορεί να αλλάζουν οι ποδοσφαιριστές, οι προπονητές και το γήπεδο – έδρα της. Αξίζει να αναφερθεί ότι οι ομάδες διαθέτουν ένα ρόστερ από 18 περίπου παίκτες, καθώς και έναν ή δύο προπονητές, ανάλογα με το συμβόλαιο συνεργασίας τους.

Μέσα από κατάλληλα ερωτήματα στη βάση, άλλα πιο σύνθετα, άλλα πιο απλά, καταφέρνουμε να συλλέγουμε στοιχεία για τους αγώνες και τα αποτελέσματά τους, τα γκολ, τις κάρτες και τα λεπτά συμμετοχής κάθε παίκτη, καθώς και τη βαθμολογία των ομάδων λαμβάνοντας υπόψη τις ποινές που ενδεχομένως έχουν υποστεί.

1. ΜΕΘΟΔΟΛΟΓΙΑ – ΒΗΜΑΤΑ Σχεδιασμου της βασησ
   1. ERD

Πρώτο βήμα για τον σχεδιασμό της βάσης, όπως διδαχθήκαμε και στα πλαίσια του μαθήματος, καθίσταται δημιουργία του μοντέλου οντοτήτων – συσχετίσεων (ERD) με βάση τις απαιτήσεις του μικρόκοσμού μας.

Σε πρώτη φάση, λοιπόν, ορίσαμε τις βασικές οντότητες που κρίναμε απαραίτητο να απαρτίζουν τη βάση μας και οι οποίες είναι οι εξής: «ΑΓΩΝΙΣΤΙΚΗ ΠΕΡΙΟΔΟΣ», «ΑΓΩΝΙΣΤΙΚΗ», «ΚΑΤΗΓΟΡΙΑ», «ΣΩΜΑΤΕΙΟ», «ΟΜΑΔΑ ΑΝΑ ΣΕΖΟΝ» (διότι, όπως προαναφέρθηκε, κάθε σεζόν κάθε σωματείο λαμβάνει μέρος στο πρωτάθλημα με ομάδα με διαφορετικά χαρακτηριστικά), «ΓΗΠΕΔΟ», «ΠΟΔΟΣΦΑΙΡΙΣΤΗΣ», «ΑΓΩΝΑΣ». Έπειτα, αποφασίσαμε πως θα ήταν χρήσιμο να δημιουργήσουμε κάποιες οντότητες που θα μας βοηθούσαν στη συνέχεια να λαμβάνουμε πληροφορίες για τα στατιστικά των παικτών, καθώς και για τη βαθμολογία κάθε ομάδας και άρα τη θέση τους στο πρωτάθλημα. Οι προαναφερθείσες οντότητες είναι οι εξής: «ΓΚΟΛ», «ΚΑΡΤΑ», «ΠΟΙΝΕΣ» (οι οποίες αφορούν συνολικά την ομάδα κι όχι κάθε παίκτη ξεχωριστά και δύνανται να επηρεάσουν τη βαθμολογία των ομάδων μέσω των πόντων ποινής). Μία ακόμα οντότητα που δημιουργήσαμε στη συνέχεια είναι η «ΣΥΜΜΕΤΟΧΗ», μέσω της οποίας μπορούμε να βλέπουμε αν, πόσο και ως τι (βασικός, αλλαγή) συμμετείχε ένας παίκτης σε έναν αγώνα. Τέλος, θέλαμε να φτιάξουμε μία οντότητα «ΑΠΟΤΕΛΕΣΜΑ», όπου θα καταγράφονται το σκορ των αγώνων, τα γκολ υπέρ και τα γκολ κατά, αλλά τελικά αποφασίσαμε ότι αυτά μπορούν να προκύψουν από ερωτήματα στη βάση.

Στη συνέχεια, καταγράψαμε και τοποθετήσαμε στο ERD τα γνωρίσματα κάθε οντότητας, ανάλογα με τα στοιχεία που θέλουμε να αποθηκεύουμε στη βάση μας. Κάποια από αυτά, λόγω της μοναδικότητας που θέλαμε να έχουν, τα ορίσαμε κλειδιά και τα υπογραμμίσαμε.

Το επόμενο και τελευταίο βήμα στη διαμόρφωση του ERD ήταν η δημιουργία των συσχετίσεων μεταξύ των οντοτήτων και ο καθορισμός των πληθικοτήτων, τα οποία υλοποιήθηκαν με βάση τις απαιτήσεις και τα όρια του μικρόκοσμου, όπως αυτά έχουν τεθεί παραπάνω.

Στην παρακάτω εικόνα φαίνεται, λοιπόν, το τελικό ERD που δημιουργήσαμε.

Εικόνα που περιέχει κύκλος, σχεδίαση

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 1: Μοντέλο οντοτήτων – συσχετίσεων της βάσης. (<https://hci.ece.upatras.gr/erdmaker/designer>)

* 1. Σχεσιακό σχήμα

Το επόμενο βήμα για τον σχεδιασμό της βάσης είναι η δημιουργία του σχεσιακού σχήματος με βάση το ERD υλοποιήσαμε στο προηγούμενο βήμα. Για τον σχεδιασμό του, λοιπόν, ακολουθήθηκε η μεθοδολογία που διδαχθήκαμε στα πλαίσια του μαθήματος.

Περιληπτικά, αρχικά δημιουργήσαμε τους πίνακες για τις κύριες οντότητες με γνωρίσματα τις ιδιότητες τους και ξένα κλειδιά τα αντίστοιχα, ενώ για πλειότιμες ιδιότητες δημιουργήσαμε νέους πίνακες (Προπονητές). Νέο πίνακα δημιουργήσαμε και για την ασθενή οντότητα «ΟΜΑΔΑ ΑΝΑ ΣΕΖΟΝ» ακολουθώντας τον κανόνα για τα ξένα κλειδιά. Στη συνέχεια, δημιουργήθηκαν οι πίνακες για συσχετίσεις τύπου Μ:Ν, όπως για παράδειγμα πίνακας «ΛΑΜΒΑΝΕΙ» (μετονομάστηκε σε «TEAM\_PUNISHMENT», όπως συνέβη και με όλους τους υπόλοιπους πίνακες, για να υπάρχει συνάφεια με τους πίνακες στη βάση), ενώ τροποποιήθηκαν ανάλογα και πίνακες που δημιουργήσαμε προηγουμένως λόγω συσχετίσεων 1:Ν.

Συνεπώς, το σχεσιακό σχήμα που δημιουργήθηκε είναι αυτό που φαίνεται στην παρακάτω εικόνα.

Εικόνα που περιέχει κείμενο, παράλληλα, στιγμιότυπο οθόνης, αριθμός

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 2: Σχεσιακό σχήμα της βάσης. (<https://150.140.186.221:1337/>)

1. μεθοδολογια – υλοποιηση της βασησ
   1. Υλοποίηση πινάκων

Το πρώτο βήμα για την υλοποίηση της βάσης είναι η δημιουργία των tables και των triggers, η οποία έγινε με την αξιοποίηση του εργαλείου **DB browser for sqlite**, που χρησιμοποιήσαμε και στο εργαστήριο του μαθήματος.

Σε πρώτη φάση, λοιπόν, δημιουργήσαμε τους πίνακες με την εντολή “CREATE TABLE”, στους οποίους συμπεριλάβαμε όλα τα γνωρίσματα, τα πρωτεύοντα και τα ξένα κλειδιά (με τα απαιτούμενα constraints), όπως αυτά προκύπτουν από το σχεσιακό σχήμα. Ταυτόχρονα, με CHECK ορίσαμε τις αναμενόμενες τιμές των γνωρισμάτων, ώστε να ελέγχονται τα δεδομένα που θα εισάγονται στη βάση και να ανήκουν σε συγκεκριμένα πεδία. Ένα παράδειγμα δημιουργίας πίνακα είναι το παρακάτω:

DROP TABLE IF EXISTS "PLAYER";

CREATE TABLE IF NOT EXISTS "PLAYER" (

"id\_player" INTEGER NOT NULL DEFAULT '0',

"name" varchar(15) NOT NULL DEFAULT '',

"lastname" varchar(15) NOT NULL DEFAULT '',

"player\_number" INTEGER NOT NULL DEFAULT '',

"birthday" date NOT NULL DEFAULT '',

"nationality" varchar(15) NOT NULL DEFAULT '',

"shirt\_number" INTEGER NOT NULL DEFAULT '' CHECK("shirt\_number" BETWEEN 1 and 99),

"height" REAL NOT NULL DEFAULT '',

"position" varchar(15) NOT NULL DEFAULT '' CHECK("position" in ("τερματοφύλακας","αμυντικός","μέσος","επιθετικός")),

PRIMARY KEY ("id\_player"));

Με τον ίδιο τρόπο δημιουργήσαμε και όλους τους υπόλοιπους πίνακες, οπότε τελικά οι πίνακες που φτιάξαμε στο DB browser for sqlite είναι συνολικά 19.

Επίσης, υλοποιήσαμε και ένα trigger με όνομα “check\_team\_categories”, ώστε όταν εισάγονται δεδομένα στη βάση σχετικά με τους αγώνες να ελέγχεται ότι αγώνες διεξάγονται μεταξύ ομάδων που αγωνίζονται στην ίδια κατηγορία. Ο κώδικας που αναπτύχθηκε για τη δημιουργία της βάσης είναι ο παρακάτω:

CREATE TRIGGER check\_team\_categories

BEFORE INSERT ON "MATCH"

FOR EACH ROW

BEGIN

SELECT RAISE(ABORT, 'Οι ομάδες δεν ανήκουν στην ίδια κατηγορία')

WHERE (SELECT l.name

FROM (TEAM\_PER\_SEASON AS tp JOIN LEAGUE AS l on tp.id\_league = l.id\_league)

WHERE tp.id\_team\_per\_season = NEW."id\_team\_home")

!=

(SELECT l1.name

FROM (TEAM\_PER\_SEASON AS tp1 JOIN LEAGUE AS l1 on tp1.id\_league = l1.id\_league)

WHERE tp1.id\_team\_per\_season = NEW."id\_team\_guest");

END;

Θα μπορούσαν να αναπτυχθούν κι άλλα triggers, ώστε να είμαστε σίγουροι για την ορθότητα των δεδομένων, όπως για παράδειγμα trigger που να ελέγχει ότι ένας παίκτης δέχεται κάρτα σε λεπτό που βρίσκεται στο παιχνίδι. Ωστόσο, επειδή στα πλαίσια της παρούσας εργασίας τέτοια θέματα τα έχουμε καλύψει προγραμματίζοντας με python τα δεδομένα που εισάγονται στη βάση και αφορούν πίνακες που παρουσιάζουν αρκετές συσχετίσεις, δεν υλοποιήσαμε άλλα triggers. Θα μπορούσε, βέβαια, αυτό να τεθεί ως ιδέα για ενδεχόμενη μελλοντική βελτίωση της βάσης μας.

* 1. Εισαγωγή δεδομένων στη βάση

Για την εισαγωγή των δεδομένων χρησιμοποιήσαμε 2 τεχνικές:

* Xρήση **ai** και συγκεκριμένα του **Microsoft Copilot**
* Προγραμματισμός με **python**

Πιο συγκεκριμένα:

* Με την χρήση του Microsoft Copilot δημιουργήσαμε τα δεδομένα που εισήχθησαν στον πίνακα για τα σωματεία (CLUB), την ομάδα ανά σεζόν **(**TEAM\_PER\_SEASON**)**, τα γήπεδα **(**ARENA**)** και τους προπονητές **(**COACH**).**

Στην εικόνα που ακολουθεί έχουμε ένα παράδειγμα χρήσης του Microsoft Copilot:

Εικόνα που περιέχει κείμενο, γραμματοσειρά, στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, έγγραφο

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα που περιέχει κείμενο, γράμμα, γραμματοσειρά, έγγραφο

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 3: Εισαγωγή δεδομένων στον πίνακα COACH με το Microsoft Copilot

Επιπρόσθετα, με χρήση του Microsoft Copilot ήταν εύκολη η εισαγωγή δεδομένων σε πίνακες με λίγες εγγραφές, όπως στους “SEASON”, “LEAGUE”, “LEAGUE\_PER\_SEASON”. Ενώ τους πίνακες “PUNISHMENT” και “TEAM\_PUNISHMENT” που αναφέρουν ποινές, όπως αφαίρεσης βαθμών από την τρέχουσα σεζόν, τους δημιουργήσαμε τυχαία με το χέρι.

* Τους πίνακες που απαιτούσαν περισσότερες εγγραφές τους υλοποιήσαμε με συναρτήσεις στην python.

Αυτό που κάναμε, αρχικά, είναι να βρούμε έναν τρόπο να «επικοινωνούμε» με την βάση μας. Να μπορούμε, δηλαδή, να διαβάζουμε στοιχεία από τους ήδη υπάρχοντες πίνακες και να γράφουμε πίσω στην βάση. Για να το πετύχουμε, αυτό, κάναμε **import** την βιβλιοθήκη **sqlite3** της python. Με τις εντολές που μας παρείχε αυτή η βιβλιοθήκη δημιουργήσαμε τις συναρτήσεις : def readSQL(*dbfilename*, *sql*, *data* = None):

db = sqlite3.connect(dbfilename)

cur = db.cursor()

if data == None:

cur.execute(sql)

else:

cur.execute(sql, data)

result = cur.fetchall()

db.close()

return result

def writeSQL(*dbfilename*, *sql*, *data* = None):

db = sqlite3.connect(dbfilename)

cur = db.cursor()

if data == None:

cur.execute(sql)

else:

cur.execute(sql, data)

# result = cur.fetchall()

db.commit()

db.close()

return 1

Με την **readSQL** αντλούμε ήδη υπάρχοντα στοιχεία από την βάση μας και με την **writeSQL** εισάγουμε δεδομένα στους πίνακες.

Στην συνέχεια, αφού βρήκαμε τρόπο να αλληλεπιδρούμε με την βάση μας, έπρεπε να εισάγουμε τα δεδομένα. Η σειρά με την οποία «γεμίζουμε» τους πίνακες, παίζει πολύ σημαντικό ρόλο διότι πολλά γνωρίσματα μερικών πινάκων συσχετίζονται με άλλα των υπόλοιπων, καθώς περιέχουν ξένα κλειδιά. Για παράδειγμα, δεν γίνεται να δημιουργήσουμε δεδομένα για τον πίνακα “PLAYER\_GOAL” χωρίς να έχουμε για τον πίνακα “PLAYER”.

Για αυτόν τον λόγο, οι πρώτοι πίνακες για τους οποίους έπρεπε να φτιάξουμε δεδομένα ήταν οι “PLAYER”και“TEAM\_PLAYER. Για να βρούμε **name** και **lastname** των ποδοσφαιριστών δημιουργήσαμε 2 μήτρες, οι οποίες περιείχαν μια πληθώρα από ελληνικά ονόματα και επίθετα. Πιο συγκεκριμένα :

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Έτσι για κάθε παίκτη, επιλέγουμε τυχαία ένα **όνομα** και ένα **επίθετο** από τους πίνακες first και last. Για την τυχαία επιλογή έχουμε δημιουργήσει την συνάρτηση **RandItem().**

def randItem(*list*):

return list[ random.randint(0, len(list)-1) ]

Για το γνώρισμα **birthday**, υλοποιήσαμε μια συνάρτηση **randDate()** η οποία μας επέστρεφε μια ημερομηνία γέννησης ανάμεσα στο 1992 και το 2006. Συγκεκριμένα :

def randDate(): # yyyy-mm-dd

    y = str(random.randint(1992, 2006))

    m = random.randint(1,12)

    if m<10:

        ms = '0'+ str(m)

else:

        ms = str(m)

    d = random.randint(1,28)

    if d<10:

        ds = '0'+ str(d)

    else:

        ds = str(d)

    return y +'-'+ ms +'-'+ ds

Για το γνώρισμα **nationality** (εθνικότητα), επιλέγουμε πάλι μια τυχαία επιλογή από τον πίνακα:

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, αριθμός

Περιγραφή που δημιουργήθηκε αυτόματα

Επειδή, όπως είναι λογικό η πλειοψηφία των παικτών είναι Έλληνες, δίνουμε αρκετές φορές την εθνικότητα ‘Ελλάδα’ για να είναι και πιο πιθανό να την επιλέξουμε.

Για το γνώρισμα **height** επιλέγουμε για κάθε έναν αριθμό ανάμεσα στο 1.45 και το 2.05.

Για το **position** και το shirt\_**number** ακολουθούμε την εξής λογική. Για τους πρώτους 15 ποδοσφαιριστές μια ομάδας, η θέση και ο αριθμός της φανέλας τους σχετίζονται. Για παράδειγμα ο αριθμός 1 και 15 μπορεί να καταχωρηθεί μόνο σε τερματοφύλακες ενώ το 2 μόνο σε αμυντικούς. Οι παίκτες από τον 16ο και μετά μπορεί να αγωνίζονται σε οποιαδήποτε θέση. Αυτό φαίνεται στο παρακάτω πίνακα :

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, γραμμή

Περιγραφή που δημιουργήθηκε αυτόματα

Από τον αριθμό 16 και μετά, η τυχαία θέση υπολογίζεται από τον πίνακα :

Εικόνα που περιέχει κείμενο, γραμματοσειρά, στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

Τέλος, κάθε παίκτης έχει ένα μοναδικό κλειδί, το id\_player το οποίο εξαρτάται από την ομάδα στην οποία αγωνίζεται.



Για παράδειγμα οι παίκτες της πρώτης ομάδας θα μπορούν να έχουν id από 40 μέχρι 59, της δεύτερης από 60 μέχρι 79 κτλ.

Συνολικά, όλα αυτά τα γνωρίσματα υπολογίζονται στην συνάρτηση **Player** η οποία παίρνει ως ορίσματα το **id της ομάδας** για την οποία δημιουργούμε τους παίκτες και το **index** από το οποίο δημιουργούμε τα id και τα shirt\_numbers.

Συνάρτηση για την δημιουργία παίκτη :

def player(*index*, *team\_id*):

    name = randItem(first)

    lastname = randItem(last)

    birthday = randDate()

    height = str(random.randint(145,204)/100)

    pos = shirt[index][1]

    if pos == "":

        pos = randItem(position)

    return {

        "id\_player": (team\_id+1) \* 20 + index,

        "name": name,

        "lastname": lastname,

        "birthday": birthday,

        "nationality": randItem(nationality),

        "shirt\_number": shirt[index][0],

        "height": height,

        "position": pos

    }

Για να δημιουργήσουμε, όμως, παίκτες πρέπει να γνωρίζουμε και την ομάδα στην οποία αγωνίζονται. Για αυτό, πρώτα πρέπει να διαβάσουμε και να αντλήσουμε από την βάση μας τα δεδομένα για τις ομάδες που αγωνίζονται κάθε σεζόν. Για αυτό θα χρησιμοποιήσουμε την συνάρτηση **readSQL** που εξηγήσαμε παραπάνω. Συγκεκριμένα :

Εικόνα που περιέχει κείμενο, γραμματοσειρά, στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

teamsSql = readSQL(database,'''

    SELECT

        tps.id\_team\_per\_season, tps.EPO\_id,

        c.name, c.city,

        s.start, s.end

        FROM TEAM\_PER\_SEASON AS tps

        LEFT JOIN CLUB c ON c.EPO\_id = tps.EPO\_id

        LEFT JOIN SEASON s ON s.id\_season = tps.id\_season

    ''')

Συνεπώς αν πάρουμε το 1ο στοιχείο του πίνακα **teamsSql** παίρνουμε τα id\_per\_season, αν πάρουμε το 5ο έχουμε την ημερομηνία έναρξης της season κτλ. Θεωρούμε ότι τα συμβόλαια των παικτών ξεκινάνε με την αρχή της σεζόν και ολοκληρώνονται στο τέλος της.

Με όλα αυτά τα στοιχεία που έχουμε συλλέξει μπορούμε να δώσουμε πλήρη δεδομένα στο πίνακες **PLAYER** και **TEAM\_PLAYER**. Όπως προαναφέραμε, θεωρούμε ότι έχουμε 36 ομάδες με 18 ποδοσφαιριστές η κάθε ομάδα. Συνολικά λοιπόν η βάση μας έχει 18 x 36 = 648 παίκτες.

Εικόνα που περιέχει κείμενο, γραμματοσειρά, στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

Τα δεδομένα τα αντλούμε με την συνάρτηση **create\_team\_players()**. Με την συνάρτηση **writeSQL** εισάγουμε τα δεδομένα.

def create\_team\_players():  # fill PLAYER and TEAM\_PLAYER tables

    for ti in range(0, number\_of\_teams):    # from ARENA

        players = []

        for i in range(0, number\_of\_players):

            plr = player(i, teamsSql[ti][0])

            writeSQL(database,'''

                INSERT INTO PLAYER (

                    "id\_player",

                    "name",

                    "lastname",

                    "birthday",

                    "nationality",

                    "shirt\_number",

                    "height",

                    "position"

                ) VALUES (?, ?, ?, ?, ?, ?, ?, ?)

            ''', (

                plr["id\_player"],

                plr["name"],

                plr["lastname"],

                plr["birthday"],

                plr["nationality"],

                plr["shirt\_number"],

                plr["height"],

                plr["position"]

            ))

            players.append(plr);

        clubs.append({

            "id\_team\_per\_season": teamsSql[ti][0],

            "name": teamsSql[ti][2],

            "players": players,

            "start\_date": teamsSql[ti][4],

            "end\_date": teamsSql[ti][5],

        })

        for p in players:

            writeSQL(database,'''

                INSERT INTO TEAM\_PLAYER (

                    id\_team\_per\_season,

                    id\_player,

                    start\_date,

                    end\_date

                    ) VALUES (?, ?, ?, ?)

                ''', (

                        teamsSql[ti][0],

                        p["id\_player"],

                        teamsSql[ti][4],

                        teamsSql[ti][5]

                )

            )

Μερικές από τις πλειάδες που δημιουργούμε στην βάση μας :

Εικόνα που περιέχει στιγμιότυπο οθόνης, κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, κατάλογος, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Στην συνέχεια, έπρεπε να φτιάξουμε τα δεδομένα για τους πίνακες **MATCHWEEK** και **MATCH**. Κάθε κατηγορία διαθέτει 12 ομάδες οπότε είναι σημαντικό να φτιάξουμε το πρόγραμμα αγώνων για όλες αυτές τις ομάδες έτσι ώστε στο τέλος του πρωταθλήματος όλες να έχουν παίξει με όλες 2 φορές (μια εντός και μία εκτός). Για αυτό δημιουργήσαμε μια συνάρτηση **create\_round\_robin\_schedule** η οποία παίρνει ως όρισμα κάποιες ομάδες και δημιουργεί το προαναφερθέν. Για παράδειγμα, στην πρώτη αγωνιστική η ομάδα 1 παίζει με την ομάδα 12, η 2 με την 11 κτλ. Ο αλγόριθμος που υπολογίζει το πρόγραμμα αγώνων φαίνεται παρακάτω.

def create\_round\_robin\_schedule(*teams*):

    """

    Δημιουργεί πρόγραμμα πρωταθλήματος με γύρους όπου κάθε αγωνιστική

    έχει τον μέγιστο αριθμό αγώνων και καμία ομάδα δεν παίζει πάνω από μία φορά.

    """

    if len(teams) % 2 != 0:

        teams.append("Ρεπό")  # Προσθήκη "Ρεπό" αν ο αριθμός των ομάδων είναι περιττός

    num\_teams = len(teams)

    schedule = []

    for round\_num in range(num\_teams - 1):

        round\_matches = []

        for i in range(num\_teams // 2):

            match = (teams[i], teams[num\_teams - 1 - i])

            round\_matches.append(match)

        schedule.append(round\_matches)

        # Περιστροφή των ομάδων για τον επόμενο γύρο

        teams = [teams[0]] + [teams[-1]] + teams[1:-1]

    return schedule

Με αυτήν την συνάρτηση υπολογίζουμε τους 11 αγώνες που θα δώσει η κάθε ομάδα. Οι υπόλοιποι 11 αγώνες τις αγωνιστικής περιόδου είναι οι ίδιοι, απλά το μόνο που αλλάζει είναι η γηπεδούχος και η φιλοξενούμενη ομάδα. Το πλήρες πρόγραμμα υπολογίζεται στην συνάρτηση **schedule\_champ :**

def create\_round\_robin\_schedule(*teams*):

    if len(teams) % 2 != 0:

        teams.append("Ρεπό")  # Προσθήκη "Ρεπό" αν ο αριθμός των ομάδων είναι περιττός

    num\_teams = len(teams)

    schedule = []

    for round\_num in range(num\_teams - 1):

        round\_matches = []

        for i in range(num\_teams // 2):

            match = (teams[i], teams[num\_teams - 1 - i])

            round\_matches.append(match)

        schedule.append(round\_matches)

        # Περιστροφή των ομάδων για τον επόμενο γύρο

        teams = [teams[0]] + [teams[-1]] + teams[1:-1]

    return schedule

Επιπλέον, όλοι οι αγώνες πραγματοποιούνται μια συγκεκριμένη ημερομηνία. Η μια αγωνιστική απέχει από την άλλη μια εβδομάδα για αυτό θεωρούμε ότι κάθε ομάδα παίζει ανά 7 ημέρες. Για να δημιουργήσουμε τις τυχαίες ημερομηνίες χρησιμοποιούμε τις συναρτήσεις :

import datetime

def days\_after(*base*, *n*):

    base\_date = datetime.datetime.strptime(base, "%Y-%m-%d")

    end\_date = base\_date + datetime.timedelta(*days*=n)

    #print(base, end\_date)

    return end\_date.strftime("%Y-%m-%d")

def matchdays\_of\_round(*base*, *round*):

    return (days\_after(base, round\*7), days\_after(base, round\*7 +3))

Επομένως, έχοντας συλλέξει όλα αυτά μπορούμε να δημιουργήσουμε δεδομένα για τους δύο πίνακες.

Η συνάρτηση που γράφει δεδομένα στους πίνακες είναι η :

def create\_schedule(*season\_id*, *league\_id*):

    writeSQL(database, '''

        DELETE FROM MATCHWEEK WHERE id\_season = ? AND id\_league = ?

        ''', (season\_id, league\_id)

    )

    min\_mw = (season\_id \* 10 + league\_id) \* 100 -1

    max\_mw = (season\_id \* 10 + league\_id) \* 100 + 99

    writeSQL(database, '''

        DELETE FROM MATCH WHERE id\_matchweek > ? AND id\_matchweek < ?

        ''', (min\_mw, max\_mw)

    )

    matchweeks = schedule\_champ(champ)

    round = 0

    for mw in matchweeks:

        round = round + 1

        id\_matchweek = (season\_id \* 10 + league\_id) \* 100 + round

        weekdates = matchdays\_of\_round(seas[season\_id][1], round)

        writeSQL(database, '''

            INSERT INTO MATCHWEEK(

                 id\_matchweek, round, id\_season, id\_league,

                 start\_date, end\_date

            ) VALUES (?, ?, ?, ?, ?, ?)

            ''',(

                id\_matchweek,

                round,

                season\_id,

                league\_id,

                weekdates[0],

                weekdates[1]

            ))

        gmi = 0

        for gm in mw:

            gmi = gmi +1

            ref0 = randItem(first)+ " "+ randItem(last);

            ref1 = randItem(first)+ " "+ randItem(last);

            ref2 = randItem(first)+ " "+ randItem(last);

            id\_match = id\_matchweek \* 10 + gmi

            writeSQL(database, '''

                INSERT INTO MATCH (id\_match, id\_matchweek, match\_date, match\_time,

                    id\_team\_home, id\_team\_guest,

                    referee, assistant\_referee1, assistant\_referee2 )

                VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)

                ''',(

                    id\_match,

                    id\_matchweek,

                    weekdates[0],

                    '16:00',

                    gm[0], gm[1],

                    ref0, ref1, ref2

                ))

Τη συνάρτηση, αυτήν, την εκτελούμε 3 φορές (1 για κάθε league (κατηγορία) με τις αντίστοιχες ομάδες που συμμετέχουν στην κατηγορία (πίνακας **champ**)) και παίρνουμε στο σύνολο :

22 (γύρους) x 3 (κατηγορίες) = 66 εγγραφές στον πίνακα **MATCHWEEK**

22 x 3 x 6 (αγώνες / αγωνιστική ) = 396 εγγραφές για τον πίνακα **MATCH**

Επίσης, τα ονόματα των διαιτητών συλλέγονται με όμοιο τρόπο με αυτά των ποδοσφαιριστών.

Σημείωση για τα **id\_match** και **id\_matchweek**:

* **id\_matchweek** είναι ένας 4-ψήφιος αριθμός που περιέχει το **id\_season** σαν πρώτο ψηφίο, το **id\_league** σαν δεύτερο ψηφίο και τον αριθμό της αγωνιστικής στα τελευταία 2 ψηφία.
* **id\_match** είναι ένας 5-ψήφιος αριθμός που περιέχει το **id\_matchweek** στα πρώτα 4 ψηφία και τον αριθμό του αγώνα στο 5ο ψηφίο.

Για παράδειγμα για το match με **id\_match** 🡪 12083 καταλαβαίνουμε ότι είναι ο 3ος αγώνας της 1ης σεζόν, της 2ης κατηγορίας, της 8ης αγωνιστικής.

Μερικά ενδεικτικά στοιχεία από τους πίνακες **MATCHWEEK** ΚΑΙ **MATCH** :

Εικόνα που περιέχει κείμενο, γραμματοσειρά, αριθμός, γραμμή

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, αριθμός

Περιγραφή που δημιουργήθηκε αυτόματα

Στην συνέχεια, έχοντας τους αγώνες, έπρεπε να βρούμε δεδομένα για τις κάρτες, τα γκολ και τη συμμετοχή των παικτών σε αυτά, δηλαδή για τους πίνακες **GOAL, PLAYER\_GOAL, CARD, PLAYER\_CARD, PARTICIPATION, PLAYER\_PARTICIPATION**.

Για τους πίνακες GOAL και CARD:

def create\_types():

    # card

    writeSQL(database, '''

            INSERT INTO CARD (id\_card, type)

            VALUES (?, ?), (?, ?);

        ''', (1, "κίτρινη", 2, "κόκκινη")

    )

    # goal types

    writeSQL(database, '''

            INSERT INTO GOAL (id\_goal, type)

            VALUES (?, ?), (?, ?);

        ''', (1, "γκολ\_υπέρ", 2, "αυτογκόλ")

    )

Το μόνο που κοιτάμε είναι το γκολ αν είναι υπέρ ή αυτογκόλ και το χρώμα της κάρτας.

**Συμβιβασμοί για τη δημιουργία δεδομένων :**

* **Όσον αφορά τη συμμετοχή των παικτών :**

Σε κάθε αγώνα, κάθε ομάδα αγωνίζεται με 11 παίκτες και έχει δικαίωμα για 3 αλλαγές, αλλά δεν είναι υποχρεωμένη να τις κάνει όλες. Επίσης, πάντα πρέπει να βρίσκεται στον αγωνιστικό χώρο ένας τερματοφύλακας. Επομένως, οι 10 από τους 11 βασικούς προκύπτουν από τους παίκτες που είναι αμυντικοί, επιθετικοί και μέσοι.

* **Όσον αφορά τις κάρτες των παικτών :**

Κάθε ομάδα μπορεί να δεχτεί μέχρι 5 κίτρινες κάρτες σε έναν αγώνα ενώ έχει 10% πιθανότητα να δεχτεί κόκκινη κάρτα. Οι κάρτες δίνονται σε τυχαίο λεπτό, στο οποίο ο συγκεκριμένος παίκτης αγωνίζεται. Επίσης, όταν ένας παίκτης δεχθεί κόκκινη κάρτα δεν μπορεί να παίξει στον επόμενο αγώνα της ομάδας του.

* **Όσον αφορά τα γκολ των παικτών :**

Κάθε γκολ έχει 10% πιθανότητα να είναι αυτογκόλ και 90% κανονικό γκολ.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Για την δημιουργία δεδομένων για συμμετοχές και κάρτες δημιουργήσαμε την συνάρτηση **participation\_and\_cards**.

def participation\_and\_cards(*list*):

…

…

Αρχικά, διαχωρίζει τους τερματοφύλακες από τους υπόλοιπους παίκτες και τους ανακατεύει έτσι ώστε σε κάθε ματς να διαλέγονται τυχαία οι βασικοί και οι αναπληρωματικοί. Επίσης, για κάθε αγώνα διαχωρίζει αυτούς που θα παίξουν σε όλο το ματς με αυτούς που είτε θα είναι βασικοί και θα βγουν αλλαγή, είτε θα μπουν ως αλλαγή. Το **noch** δείχνει τον αριθμό των αλλαγών που θα κάνει ένας προπονητής και είναι ένας τυχαίος αριθμός από 1 έως 3.

…

for (id, pos) in list:

        if pos == "τερματοφύλακας":

            keeper.append(id)

        else:

            other.append(id)

    random.shuffle(keeper)

    random.shuffle(other)

    play\_all\_match = other[0 : (basikoi - noch)]

    play\_all\_match.append(keeper[0])

    changes = other[(basikoi-noch) : ((basikoi-noch) + noch\*2)]

…

* **Για τους παίκτες που παίζουν όλο τον αγώνα :**

Αγωνίζονται από minute\_in = 0 έως minute\_out = 90 εκτός εάν έχουν δεχτεί κόκκινη κάρτα, όπου το minute\_out ισούται με το λεπτό στο οποίο δέχτηκαν την κόκκινη.

* **Για τους παίκτες που είναι ή θα βγουν αλλαγή :**

Για κάθε έναν παίκτη που βγαίνει αλλαγή σε ένα λεπτό υπάρχει ένας άλλος που εισέρχεται στον αγωνιστικό χώρο στο ίδιο λεπτό.

for plr in play\_all\_match:

        if has\_red\_card and plr == play\_all\_match[0]:

            red\_card\_min = random.randint(5,85)

            participation.append(

                (plr, 0, red\_card\_min )

            )

            cards.append(

                (plr, 2, red\_card\_min)

            )

        else:

            participation.append(

                (plr, 0, 90)

            )

    for i in range(0, noch):

        min = random.randint(40, 85)

        participation.append(

            ( changes[i] , 0, min )

        )

        participation.append(

            ( changes[i + noch], min, 90)

        )

    for i in range(0, random.randint(0, max\_yellow)):

        (id, min\_in, min\_o) = participation[random.randint(0, len(participation)-1)]

        if not id in have\_yellow:

            have\_yellow.append(id)

            cards.append(

                (id, 1, random.randint(min\_in, min\_o))

            )

    return {

        "keep": keeper,

        "other": other,

        "part\_tbl": participation,

        "cards": cards

    }

Επίσης, στο τέλος η συνάρτηση επιστρέφει 4 πίνακες. Έναν που δείχνει τους τερματοφύλακες της ομάδας (**keep**) , έναν που δείχνει τους υπόλοιπους παίκτες που έχουν δικαίωμα συμμετοχής στον αγώνα (**other**), έναν που δείχνει τους παίκτες που έχουν συμμετοχή καθώς και τα λεπτά συμμετοχής τους (**part\_tbl**) και έναν που δείχνει τους παίκτες που έχουν δεχτεί κάρτα (**cards**).

Για την δημιουργία δεδομένων για τα γκολ δημιουργήσαμε την συνάρτηση **goal\_achivers** :

Η συνάρτηση, αυτή, επιστρέφει έναν πίνακα **goals** που περιέχει τα id των ποδοσφαιριστών που σκοράρουν, τον τύπο του γκολ τους και το λεπτό που το πετυχαίνουν.

* **Για τους παίκτες που πετυχαίνουν τα γκολ στους αγώνες :**

Μας ενδιαφέρει να έχουν συμμετοχή στο τυχαίο λεπτό που θα πετύχουν το γκολ (να είναι δηλαδή ένα λεπτό ανάμεσα στο **minute\_in** και το **minute\_out** του παίκτη) . Επίσης, τα γκολ της γηπεδούχου ομάδας (**goalsHome**) τα πετυχαίνουν είτε παίκτες της ίδιας της ομάδας και είναι γκολ υπέρ, είτε παίκτες της φιλοξενούμενης ομάδας και είναι αυτογκόλ. Αντίστοιχα και για τα γκολ της φιλοξενούμενης ομάδας (**goalsGuest**).

Η υλοποίηση της συνάρτησης **goal\_achivers** :

def goal\_achivers(*home\_participation*, *guest\_participation*, *goalsHome*, *goalsGuest*):

    goals = []

    goalMinutes = []

    for i in range(0, goalsHome):

        is\_own\_goal = False

        if random.randint(0,100) < owngoal\_pc:

            is\_own\_goal = True

        if is\_own\_goal:

     (id, min\_in, min\_out) = guest\_participation[random.randint(0, len(guest\_participation)-1)]

            minute = rand\_int\_in\_range(min\_in, min\_out, goalMinutes)

            goals.append(

                (id, 2, minute)

            )

            goalMinutes.append(minute)

        else:

    (id, min\_in, min\_out) = home\_participation[random.randint(0, len(home\_participation)-1)]

            minute = rand\_int\_in\_range(min\_in, min\_out, goalMinutes)

            goals.append(

                (id, 1, minute)

            )

            goalMinutes.append(minute)

    for i in range(0, goalsGuest):

        is\_own\_goal = False

        if random.randint(0,100) < owngoal\_pc:

            is\_own\_goal = True

        if is\_own\_goal:

   (id, min\_in, min\_out) = home\_participation[random.randint(0, len(home\_participation)-1)]

            minute = rand\_int\_in\_range(min\_in, min\_out, goalMinutes)

            goals.append(

                (id, 2, minute)

            )

            goalMinutes.append(minute)

        else:

            (id, min\_in, min\_out) = guest\_participation[random.randint(0, len(guest\_participation)-1)]

            minute = rand\_int\_in\_range(min\_in, min\_out, goalMinutes)

            goals.append(

                (id, 1, minute)

            )

            goalMinutes.append(minute)

    return goals

Αυτές, λοιπόν, τις δύο συναρτήσεις που είδαμε παραπάνω θα τις χρησιμοποιήσουμε για να δημιουργήσουμε τα στατιστικά δεδομένα για τους ποδοσφαιριστές.

Αυτό που πρέπει να κάνουμε αρχικά είναι να διαβάσουμε από την βάση μας όλους τους ποδοσφαιριστές όλων των ομάδων που δημιουργήσαμε. Για να το κάνουμε, αυτό, θα χρησιμοποιήσουμε πάλι την συνάρτηση **ReadSQL** που έχουμε προαναφέρει. **Όμως**, από τον 2ο γύρο και μετά δεν θα πρέπει να συλλέγουμε όλους τους παίκτες αλλά μόνο όσους έχουν δικαίωμα συμμετοχής στον εν λόγω αγώνα, δηλαδή, αυτούς που δεν έχουν πάρει κόκκινη κάρτα στην προηγούμενη αγωνιστική. Για αυτό πρέπει να συντάσσουμε δύο ερωτήματα SQL, ένα για την πρώτη αγωνιστική και ένα για όλες τις υπόλοιπες. Αυτά τα ερωτήματα υλοποιούνται στην παρακάτω συνάρτηση. Παρατηρούμε ότι στο δεύτερο query χρησιμοποιούμε ένα υποερώτημα για να αφαιρέσουμε τους τιμωρημένους ποδοσφαιριστές.

def playersOfTeam(*id\_tps*, *round*):  # id\_team\_per\_season

    validPlayersSQL\_Round1 = '''

        SELECT

            tp.id\_player, p.position

        FROM TEAM\_PLAYER tp

        LEFT JOIN PLAYER p ON p.id\_player = tp.id\_player

        WHERE tp.id\_team\_per\_season = ?

    '''

    validPlayersSQL = '''

        SELECT

            tp.id\_player, p.position

        FROM TEAM\_PLAYER tp

        LEFT JOIN PLAYER p ON p.id\_player = tp.id\_player

        WHERE tp.id\_team\_per\_season = ?

        AND tp.id\_player NOT IN ( -- players witn red card in the prev round

            SELECT tp.id\_player

            FROM TEAM\_PLAYER tp

            LEFT JOIN PLAYER\_PARTICIPATION pp ON pp.id\_player = tp.id\_player

            LEFT JOIN MATCH m ON m.id\_match = pp.id\_match

            LEFT JOIN MATCHWEEK mw ON mw.id\_matchweek = m.id\_matchweek

            LEFT JOIN PLAYER\_CARD pc ON pc.id\_player = tp.id\_player AND pc.id\_match = m.id\_match

            WHERE (m.id\_team\_home = ? OR m.id\_team\_guest = ?)

            AND mw.round = ?    -- (round - 1)

            AND pc.id\_card = ?  -- type = 2 = red-card

        )

    '''

    if round == 1:

        return readSQL(database, validPlayersSQL\_Round1, (id\_tps, ))

    else:

        return readSQL(database, validPlayersSQL, (id\_tps, id\_tps, id\_tps, round-1, 2))

Συνεπώς, για κάθε ομάδα και για κάθε αγωνιστική μπορούμε να αντλούμε τους διαθέσιμους ποδοσφαιριστές. Επιπλέον, κάθε αγώνας έχει ένα σκορ το οποίο χρειάζεται να ξέρουμε για να δημιουργήσουμε τους σκόρερ για κάθε ομάδα. Αυτό υπολογίζεται τυχαία από τους πίνακες:

Εικόνα που περιέχει γραμματοσειρά, κείμενο, στιγμιότυπο οθόνης, γραφικά

Περιγραφή που δημιουργήθηκε αυτόματα

Παρατηρούμε ότι η εντός έδρας ομάδα έχει παραπάνω πιθανότητες για νίκη από ότι η εκτός έδρας κάτι που είναι και λογικό σε ένα πραγματικό πρωτάθλημα.

Για την εισαγωγή των δεδομένων στους πίνακες :

def run\_matches(*season*, *league*, *round*):

    mw\_id = (season \* 10 + league) \* 100 + round

    print('Matchweek:', mw\_id)

    roundMatches = readSQL(database, '''

        SELECT id\_team\_home, id\_team\_guest

        FROM MATCH WHERE id\_matchweek = ?

        ''', (mw\_id, ))

    idxi = 0

    for match in roundMatches:

        homePlayers = playersOfTeam(match[0],round)

        guestPlayers = playersOfTeam(match[1],round)

        goalsHome = randItem(goalH)

        goalsGuest = randItem(goalG)

        idxi = idxi + 1

        id\_match = mw\_id \* 10 + idxi

        home\_p\_and\_cards = participation\_and\_cards(homePlayers)

        away\_p\_and\_cards = participation\_and\_cards(guestPlayers)

        goals\_per\_match = goal\_achivers(home\_p\_and\_cards["part\_tbl"], away\_p\_and\_cards["part\_tbl"], goalsHome, goalsGuest)

Παρατηρούμε ότι αυτή η συνάρτηση δέχεται ως ορίσματα το id της season, το id της κατηγορίας και το round και υπολογίζει τα στατιστικά στοιχεία για όλα τα ματς μιας συγκεκριμένης αγωνιστικής.

Το μόνο που μένει να γίνει είναι να γραφτούν τα δεδομένα στην βάση. Για να γραφτούν θα χρησιμοποιούμε την συνάρτηση **writeSQL()**.

* Σημείωση για το **id\_participation** :

Είναι ένας αριθμός 8 ψηφιών όπου τα 5 πρώτα ψηφία δείχνουν το **id\_match** και τα τελευταία 3 το **id\_player** που έχει συμμετοχή.

**Παράδειγμα**: id\_participation 🡪 11021041 : O παίχτης με id 41 έχει συμμετοχή στον αγώνα με id 11021, δηλαδή τον 1ο αγώνα της 2ης αγωνιστικής της πρώτης κατηγορίας.

Για τους πίνακες **PLAYER\_CARD, PARTICIPATION, PLAYER\_PARTICIPATION:**

for i in range(0,len(home\_p\_and\_cards["part\_tbl"])) :

            id\_p = home\_p\_and\_cards["part\_tbl"][i][0]

            id\_participation = id\_match \* 1000 + id\_p

            writeSQL(database, '''

                INSERT INTO PARTICIPATION (id\_participation,minute\_in,minute\_out

                     )

                VALUES (?, ?, ?)

                ''',(

                    id\_participation,

                    home\_p\_and\_cards["part\_tbl"][i][1],

                    home\_p\_and\_cards["part\_tbl"][i][2]

                ))

            writeSQL(database, '''

INSERT INTO PLAYER\_PARTICIPATION id\_match,id\_player,id\_participation

                     )

                VALUES (?, ?, ?)

                ''',(

                    id\_match,

                    id\_p,

                    id\_participation

                ))

        for i in range(0,len(home\_p\_and\_cards["cards"])) :

            writeSQL(database, '''

                INSERT INTO PLAYER\_CARD (id\_match,id\_player,id\_card,minute

                     )

                VALUES (?, ?, ?, ?)

                ''',(

                    id\_match,

                    home\_p\_and\_cards["cards"][i][0],

                    home\_p\_and\_cards["cards"][i][1],

                    home\_p\_and\_cards["cards"][i][2]

                ))

Αντίστοιχα και για την εκτός έδρας ομάδα ( 2 ίδια loops με **away\_p\_and\_cards** πίνακα)

Για τον πίνακα **PLAYER\_GOAL:**

for i in range(0,len(goals\_per\_match)) :

            writeSQL(database, '''

                INSERT INTO PLAYER\_GOAL (id\_match,id\_player,id\_goal,minute

                     )

                VALUES (?, ?, ?, ?)

                ''',(

                    id\_match,

                    goals\_per\_match[i][0],

                    goals\_per\_match[i][1],

                    goals\_per\_match[i][2]

                ))

Τέλος για να μην τρέξουμε αυτήν την συνάρτηση για 3(κατηγορίες) x 22 (rounds) = 66 φορές δημιουργούμε την συνάρτηση, την οποία εκτελούμε μόνο τρεις φορές, μια για κάθε κατηγορία.

def run\_matchweeks(*season*, *league*):

    for i in range(1,23):

        run\_matches(season, league, i)

        print('αγωνιστική :', i)

Και με αυτόν τον τρόπο, δημιουργούμε μια βάση με πάρα πολλά δεδομένα. Συγκεκριμένα, στον πίνακα **PARTICIPATION, PLAYER\_PARTICIPATION** έχουμεπάνω από **9000 δεδομένα** και στους πίνακες **PLAYER\_CARD** και **PLAYER\_GOAL** πάνω από **1500**.

**Όλα αυτά τα αρχεία της python υπάρχουν και σε αρχείο zip για να μπορούν να μελετηθούν αναλυτικά.**

1. ΜΕΘΟΔΟΛΟΓΙΑ – ΕΝΤΟΛΕΣ CRUD (CREATE-READ-UPDATE-DELETE)

Εφόσον δημιουργήσαμε τη βάση μας, διατυπώσαμε κάποιες εντολές SQL που επιτρέπουν αλλαγή στο περιεχόμενό της. Αρχικά, υλοποιήσαμε τις συναρτήσεις CRUD (CREATE-READ-UPDATE-DELETE). Οι συναρτήσεις θα δέχονται ως είσοδο ένα λεξικό και θα επιστρέφουν το SQL ερώτημα που θα πρέπει να εφαρμόσουμε στην βάση μας, για να μπορούμε να κάνουμε τις απαραίτητες τροποποιήσεις.

Συνάρτηση για **insert(create)** :

def insert(*query*):

    table, field, value = itemgetter('table', 'field', 'value')(query)

    sql = f"INSERT INTO \"{table}\" "

    fList = ', '.join(field)

    sql += f"({fList})"

    valList = ', '.join(value)

    sql += f"\nVALUES ({valList})"

    return sql

Δέχεται ως είσοδο ένα λεξικό με 3 κλειδιά. Το table από το οποίο αντλούμε τον πίνακα στον οποίο θέλουμε να εισάγουμε τα δεδομένα, το field από το οποίο παίρνουμε τα γνωρίσματα στα οποία θέλουμε να δώσουμε νέες τιμές και το value το οποίο εμπεριέχει τις νέες πλειάδες που θέλουμε να εισάγουμε στην βάση μας. Ένα παράδειγμα, για κατανόηση της συνάρτησης insert :

q = {

    "table": 'USER',

    "field": [

        'username', 'password'

    ],

    "value": [

        '"Andreas"', '"0000-1-2-3-4"'

    ]

}

print(insert(q))

Το οποίο επιστρέφει :



Με την ίδια λογική δημιουργούμε και τις συναρτήσεις **read, update, delete**.

Συνάρτηση για **read** :

def read(*query*):

    table, field, where, order = itemgetter('table', 'field', 'where', 'order')(query)

    fList = "\*"

    if field != None:

        fList = ', '.join(field)

    sql = f"SELECT {fList}\n"

    sql += f"FROM {table}\n"

    whereStatment = ""

    if where != None:

        whereItems = []

        for (fname, operator, val) in where:

            whereItems.append(f"{fname} {operator} {val}")

        whereStatment = f"WHERE {' AND '.join(whereItems)}"

    # order-by is not implemented

    sql += whereStatment

    return sql

Δέχεται ως είσοδο ένα λεξικό με 4 κλειδιά. Το table από το οποίο παίρνουμε τον πίνακα, το field από το οποίο επιλέγουμε τα γνωρίσματα, το where το οποίο εμπεριέχει τις συνθήκες με τις οποίες θα επιλέξουμε τις πλειάδες και το order για την σειρά εμφάνισης τους (δεν καλύπτεται στην συνάρτηση). Έτσι δημιουργούμε ένα ερώτημα της μορφής :

**SELECT … FROM … WHERE <συνθήκη1> AND <συνθήκη2>.**

Συνάρτηση για **update** :

def update( *query* ):

    table, sets, where = itemgetter('table', 'set', 'where')(query)

    sql = f"UPDATE \"{table}\"\nSET "

    setStatements = []

    for (f, v) in sets:

        setStatements.append(f"{f} = {v}")

    sql += ', '.join(setStatements)

    whereStatment = ""

    if where != None:

        whereItems = []

        for (fname, operator, val) in where:

            whereItems.append(f"{fname} {operator} {val}")

        whereStatment = f"\nWHERE {' AND '.join(whereItems)}"

    sql += whereStatment

    return sql

Δέχεται ως είσοδο ένα λεξικό με 3 κλειδιά. Το table από το οποίο αντλούμε τον πίνακα τον οποίο θέλουμε να τροποποιήσουμε, το set από το οποίο παίρνουμε τα γνωρίσματα τα οποία θέλουμε να αλλάξουν και τις νέες τιμές τους και το where το οποίο εμπεριέχει τις συνθήκες με τις οποίες θα επιλέξουμε ποιες πλειάδες θέλουμε διαφοροποιήσουμε. Έτσι δημιουργούμε ένα ερώτημα της μορφής :

**UPDATE … SET <γνώρισμα> = <νέα τιμή> WHERE <συνθήκη1> AND <συνθήκη2>.**

Συνάρτηση για **delete** :

def delete(*query*):

    table, where = itemgetter('table', 'where')(query)

    sql = f"DELETE FROM \"{table}\""

    whereStatment = ""

    if where != None:

        whereItems = []

        for (fname, operator, val) in where:

            whereItems.append(f"{fname} {operator} {val}")

        whereStatment = f"\nWHERE {' AND '.join(whereItems)}"

    sql += whereStatment

    return sql

Δέχεται ως είσοδο ένα λεξικό με 2 κλειδιά. Το table από το οποίο αντλούμε τον πίνακα τον οποίο θέλουμε να διαγράψουμε κάποιο στοιχείο και το where το οποίο εμπεριέχει τις συνθήκες με τις οποίες θα επιλέξουμε ποιες πλειάδες θα διαγράψουμε. Έτσι δημιουργούμε ένα ερώτημα της μορφής :

**DELETE FROM … WHERE <συνθήκη1> AND <συνθήκη2>.**

Για να ενσωματώσουμε όλες αυτές τις λειτουργίες, θεωρούμε ότι στην εφαρμογή μας έχουμε έναν **Admin**. Αυτός με την σειρά του μπορεί να δημιουργεί **(insert)** κάποιους **roster\_admin** οι οποίοι είναι υπεύθυνοι για την εγγραφή νέων παικτών στην βάση και την εισαγωγή σε ομάδα. Επίσης, ο **Admin** θα έχει την δυνατότητα να διαγράφει κάποιον **roster\_admin (delete)**. Επιπλέον, κάθε **Admin** και **roster\_admin** θα έχουν ένα μοναδικό username και ένα password, το οποίο θα μπορούν να το αλλάξουν ανά πάσα στιγμή **(update)**, με τo οποίo θα μπορούν να κάνουν login στην εφαρμογή. Τέλος, όπως και οι απλοί users, θα μπορούν να διαβάσουν **(read)** όλη την θεματολογία που παρέχει η εφαρμογή.

Έτσι καλύπτονται όλων των ειδών αλλαγές που μπορούμε να κάνουμε στην βάση μας.

Παρακάτω θα δούμε ένα παράδειγμα στο οποίο περιγράφονται όλα αυτά ( **Πιο αναλυτικά στην περιγραφή της γραφικής διεπαφής** ):

Αρχικά έχουμε δημιουργήσει στην βάση μας έναν πίνακα USER και έχουμε προσθέσει τον admin της εφαρμογής.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, αριθμός

Περιγραφή που δημιουργήθηκε αυτόματα

Μενού επιλογών απλού χρήστη (user) :

Εικόνα που περιέχει κείμενο, γραμματοσειρά, στιγμιότυπο οθόνης, σχεδίαση

Περιγραφή που δημιουργήθηκε αυτόματα

Login ως Admin :

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Παρατηρούμε και όλο το μενού επιλογών που έχει ένας admin.

Επιλογή 21 για δημιουργία roster\_admin :

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Με το που δημιουργηθεί ο roster\_admin ενημερώνεται η βάση :

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, αριθμός

Περιγραφή που δημιουργήθηκε αυτόματα

Τώρα αν κάνουμε login στην εφαρμογή ως roster\_admin, θα δούμε ένα παρόμοιο μενού επιλογών στο οποίο το μόνο που έχει αφαιρεθεί είναι η δυνατότητα προσθήκης και διαγραφής κάποιου χρήστη.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Επιλογή 20 για αλλαγή κωδικού. Ο νέος κωδικός ενημερώνεται στην βάση :

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, αριθμός

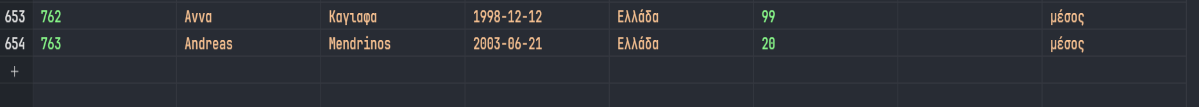
Περιγραφή που δημιουργήθηκε αυτόματα

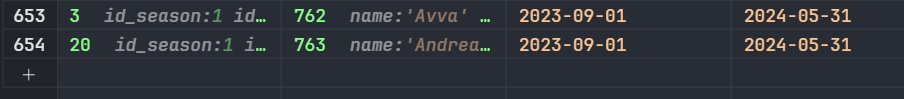
Επιλογή 10 για την δημιουργία παίκτη και την εγγραφή σε ομάδα. Συμπληρώνουμε όλα τα γνωρίσματα του παίκτη, καθώς και την ομάδα στην οποία πρόκειται να αγωνιστεί. Αυτό που χρειάζεται να προσέξουμε είναι να μην του δώσουμε κάποιο αριθμό φανέλας που έχει κάποιος συμπαίκτης του.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Και ο νέος παίκτης εισάγεται στην βάση στους πίνακες PLAYER και TEAM\_PLAYER:





Τέλος με Logout, βλέπουμε τις αρχικές επιλογές, δηλαδή αυτές που έχει ένας απλός χρήστης.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Με Exit κλείνουμε την εφαρμογή.

Όλα αυτά όπως προαναφέρθηκε, έχουν ενσωματωθεί στο γραφικό περιβάλλον.

1. ΜΕΘΟΔΟΛΟΓΙΑ – ΕΡΩΤΗΜΑΤΑ ΣΤΗ ΒΑΣΗ

Εφόσον δημιουργήσαμε τη βάση μας, διατυπώσαμε σε sqlite ερωτήματα προς αυτή για να λάβουμε διάφορα στατιστικά στοιχεία για τους παίκτες και τις ομάδες, πληροφορίες για τους αγώνες, τα αποτελέσματα και τη βαθμολογία. Τα ερωτήματα, αυτά, διατυπώθηκαν αρχικά στο **DB Browser for sqlite**, ώστε να ελέγξουμε τη σωστή λειτουργία τους και στη συνέχεια τα συμπεριλάβαμε στο πρόγραμμα σε python που γράψαμε για να δημιουργήσουμε την εφαρμογή μας. Πιο συγκεκριμένα, υλοποιήσαμε τα παρακάτω ερωτήματα:

* Για κάθε ομάδα θέλουμε όνομα ομάδας, γήπεδο, αριθμός ποδοσφαιριστών, κατηγορία, προπονητές:

SELECT t.id\_team\_per\_season, cl.name AS team\_name, a.name AS arena\_name, l.name,(SELECT COUNT(DISTINCT tp\_inner.id\_player)

FROM TEAM\_PLAYER AS tp\_inner

WHERE tp\_inner.id\_team\_per\_season = t.id\_team\_per\_season) AS player\_count,GROUP\_CONCAT(c.coach\_info, ', ') AS coaches

FROM ((((TEAM\_PER\_SEASON AS t JOIN CLUB as cl on t.EPO\_id = cl.EPO\_id) JOIN LEAGUE as l on l.id\_league = t.id\_league) JOIN ARENA AS a ON t.id\_arena = a.id\_arena) JOIN (SELECT DISTINCT id\_team\_per\_season, name || ' ' || lastname || ' (' || start\_date || ' - ' || IFNULL(end\_date, 'Present') || ')' AS coach\_info FROM COACH) AS c ON t.id\_team\_per\_season = c.id\_team\_per\_season)

WHERE player\_count <> 0

GROUP BY t.id\_team\_per\_season

ORDER BY t.id\_team\_per\_season;

* Ρόστερ ομάδων, δηλαδή για κάθε ομάδα θέλουμε ονόματα ποδοσφαιριστών, έτος γέννησης, θέση:

SELECT cl.name, p.id\_player, p.name, p.lastname, strftime('%Y', p.birthday) as year\_of\_birth, p.position

FROM (((TEAM\_PER\_SEASON as t JOIN CLUB as cl on cl.EPO\_id = t.EPO\_id) join TEAM\_PLAYER as tp on t.id\_team\_per\_season = tp.id\_team\_per\_season) JOIN PLAYER as p on tp.id\_player = p.id\_player)

ORDER BY cl.name, p.lastname;

* + Θέλουμε το αποτέλεσμα των αγώνων, δηλαδή το id του αγώνα, τα id των ομάδων που αγωνίζονται, καθώς και τα γκολ υπέρ και γκολ κατά (αυτογκόλ) που βάζει κάθε ομάδα. Στο σημείο, αυτό, αποφασίσαμε το συγκεκριμένο ερώτημα να μετατραπεί σε View, ώστε τα αποτελέσματά του να μπορούμε να τα αξιοποιήσουμε και σε επόμενα ερωτήματα και να μην χρειαστεί να τα υπολογίσουμε ξανά. Οπότε, ο κώδικας που αναπτύχθηκε είναι ο ακόλουθος:

CREATE VIEW MATCH\_RESULT AS

SELECT m.id\_match, m.id\_team\_home, m.id\_team\_guest,COALESCE((SELECT count(pg.id\_goal) FROM ((PLAYER\_GOAL as pg JOIN GOAL as g on pg.id\_goal = g.id\_goal) JOIN TEAM\_PLAYER as tp on tp.id\_player = pg.id\_player)

WHERE tp.id\_team\_per\_season = m.id\_team\_home AND g.type = "γκολ\_υπέρ" AND pg.id\_match = m.id\_match

GROUP BY pg.id\_match),0) as goal\_home\_over, COALESCE((SELECT count(pg1.id\_goal)

FROM ((PLAYER\_GOAL as pg1 JOIN GOAL as g1 on pg1.id\_goal = g1.id\_goal) JOIN TEAM\_PLAYER as tp1 on tp1.id\_player = pg1.id\_player)

WHERE tp1.id\_team\_per\_season = m.id\_team\_home AND g1.type = "αυτογκόλ" AND pg1.id\_match = m.id\_match

GROUP BY pg1.id\_match),0) as goal\_home\_against, COALESCE((SELECT count(pg2.id\_goal)

FROM ((PLAYER\_GOAL as pg2 JOIN GOAL as g2 on pg2.id\_goal = g2.id\_goal) JOIN TEAM\_PLAYER as tp2 on tp2.id\_player = pg2.id\_player)

WHERE tp2.id\_team\_per\_season = m.id\_team\_guest AND g2.type = "γκολ\_υπέρ" AND pg2.id\_match = m.id\_match

GROUP BY pg2.id\_match),0) as goal\_guest\_over, COALESCE((SELECT count(pg3.id\_goal)

FROM ((PLAYER\_GOAL as pg3 JOIN GOAL as g3 on pg3.id\_goal = g3.id\_goal) JOIN TEAM\_PLAYER as tp3 on tp3.id\_player = pg3.id\_player)

WHERE tp3.id\_team\_per\_season = m.id\_team\_guest AND g3.type = "αυτογκόλ" AND pg3.id\_match = m.id\_match

GROUP BY pg3.id\_match),0) as goal\_guest\_against

FROM MATCH as m

* + Θέλουμε να δημιουργήσουμε το πρόγραμμα των αγώνων. Πιο συγκεκριμένα, για τους αγώνες σε όλες τις αγωνιστικές όλων των κατηγοριών θέλουμε τις εκάστοτε ομάδες που αγωνίζονται, το γήπεδο που διεξάγεται ο αγώνας (είναι έδρα μίας από τις δύο ομάδες), ημέρα και ώρα αγώνα, το σκορ και τους διαιτητές. Για να υπολογίσουμε το σκορ, μετράμε όλα τα γκολ υπέρ και αυτογκόλ κάθε μίας από τις δύο ομάδες, τα οποία αντλούμε από το View που δημιουργήσαμε στο προηγούμενο ερώτημα. Ο κώδικας που αναπτύχθηκε είναι ο παρακάτω:

SELECT l.name, ml.round, cl1.name as team\_home, cl2.name as team\_guest, a.name as arena\_name, m.match\_time, m.match\_date, ((mr.goal\_home\_over + mr.goal\_guest\_against) || ' - ' || (mr.goal\_guest\_over + mr.goal\_home\_against)) as score, m.referee, m.assistant\_referee1, m.assistant\_referee2

FROM ((((((((MATCH as m JOIN MATCHWEEK as ml on m.id\_matchweek = ml.id\_matchweek)JOIN league as l on ml.id\_league = l.id\_league) JOIN TEAM\_PER\_SEASON as t on t.id\_team\_per\_season = m.id\_team\_home)JOIN CLUB as cl1 on cl1.EPO\_id = t.EPO\_id)join TEAM\_PER\_SEASON as t1 on t1.id\_team\_per\_season = m.id\_team\_guest)JOIN CLUB as cl2 on cl2.EPO\_id = t1.EPO\_id) JOIN ARENA as a on a.id\_arena = t.id\_arena) JOIN MATCH\_RESULT as mr on mr.id\_match = m.id\_match);

* + Θέλουμε να καταμετρούμε τις ποινές. Πιο συγκεκριμένα, για όσες ομάδες έχουν δεχτεί ποινή να φαίνεται το όνομα και το EPO\_id τους, η αιτία της ποινής, η διάρκειά της (είναι καθαρά θεωρητική στη βάση μας και ουσιαστικά σχετίζεται με το χρονικό διάστημα που η ομάδα θα αγωνίζεται κεκλεισμένων των θυρών, το οποίο όμως εμείς εδώ δεν εξετάζουμε), το χρηματικό πρόστιμο, οι βαθμοί ποινής (που αφαιρούνται από τη βαθμολογία της ομάδας) και η ημερομηνία που επιβλήθηκε η εκάστοτε ποινή. Ο κώδικας που αναπτύχθηκε είναι ο παρακάτω:

SELECT cl.name, cl.EPO\_id, p.reason, p.duration, p.money\_penalty, p.points\_penalty, tp.punishment\_date

FROM((((TEAM\_PER\_SEASON as t JOIN CLUB as cl on t.EPO\_id = cl.EPO\_id)JOIN TEAM\_PUNISHMENT as tp on tp.id\_team\_per\_season = t.id\_team\_per\_season)JOIN PUNISHMENT as p on p.id\_punishment = tp.id\_punishment)JOIN LEAGUE as l on l.id\_league = t.id\_league)

ORDER BY cl.name;

* + Θέλουμε να παίρνουμε σε κάθε κατηγορία ποιες ομάδες ανήκουν. Το ερώτημα αυτό, ωστόσο, δεν το συμπεριλάβαμε στην εφαρμογή μας, διότι όλες τις πληροφορίες για τις ομάδες τις παίρνουμε έτσι κι αλλιώς ανά κατηγορία. Ωστόσο, ο κώδικας που αναπτύχθηκε είναι ο παρακάτω:

SELECT l.name, GROUP\_CONCAT(' (' || t.id\_team\_per\_season || ' , ' || t.EPO\_id || ' , ' || cl.name || ')') as teams

FROM ((TEAM\_PER\_SEASON as t join CLUB as cl on t.EPO\_id = cl.EPO\_id) JOIN LEAGUE as l on t.id\_league = l.id\_league)

GROUP BY l.name

* + Θέλουμε τη βαθμολογία των ομάδων ανά κατηγορία και σεζόν. Πιο συγκεκριμένα, θέλουμε το id και το όνομα των ομάδων, τους αγώνες στους οποίους έχει λάβει μέρος κάθε ομάδα, τους πόντους της, τις νίκες, τις ήττες, τις ισοπαλίες και τα συνολικά γκολ υπέρ και αυτογκόλ που έχει σημειώσει στους αγώνες που έχει συμμετάσχει. Αξίζει εδώ να σημειωθεί ότι για κάθε νίκη έχουμε 3 πόντους, για κάθε ισοπαλία έχουμε 1 πόντο και για κάθε ήττα κανέναν, όπως συμβαίνει και σε ένα πραγματικό τοπικό πρωτάθλημα ποδοσφαίρου. Στο σημείο, αυτό, αποφασίσαμε το συγκεκριμένο ερώτημα να μετατραπεί σε View, ώστε τα αποτελέσματά του να μπορούμε δυνητικά να τα αξιοποιήσουμε και σε επόμενα ερωτήματα και να μην χρειαστεί να τα υπολογίσουμε ξανά. Ο κώδικας, λοιπόν, που αναπτύχθηκε είναι ο ακόλουθος:

CREATE VIEW RANKING AS

WITH MatchStats AS (

SELECT

t.id\_team\_per\_season AS id,

-- Νίκες

SUM(CASE

WHEN mr.id\_team\_home = t.id\_team\_per\_season

AND (mr.goal\_home\_over + mr.goal\_guest\_against) > (mr.goal\_guest\_over + mr.goal\_home\_against) THEN 1

WHEN mr.id\_team\_guest = t.id\_team\_per\_season

AND (mr.goal\_home\_over + mr.goal\_guest\_against) < (mr.goal\_guest\_over + mr.goal\_home\_against) THEN 1

ELSE 0

END) AS win,

-- Ήττες

SUM(CASE

WHEN mr.id\_team\_home = t.id\_team\_per\_season

AND (mr.goal\_home\_over + mr.goal\_guest\_against) < (mr.goal\_guest\_over + mr.goal\_home\_against) THEN 1

WHEN mr.id\_team\_guest = t.id\_team\_per\_season

AND (mr.goal\_home\_over + mr.goal\_guest\_against) > (mr.goal\_guest\_over + mr.goal\_home\_against) THEN 1

ELSE 0

END) AS defeat,

-- Ισοπαλίες

SUM(CASE

WHEN (mr.id\_team\_home = t.id\_team\_per\_season OR mr.id\_team\_guest = t.id\_team\_per\_season)

AND (mr.goal\_home\_over + mr.goal\_guest\_against) = (mr.goal\_guest\_over + mr.goal\_home\_against) THEN 1

ELSE 0

END) AS draw,

-- Σύνολο Αγώνων

COUNT(mr.id\_match) AS matches,

-- Γκολ Υπέρ

SUM(CASE

WHEN mr.id\_team\_home = t.id\_team\_per\_season THEN (mr.goal\_home\_over + mr.goal\_guest\_against)

WHEN mr.id\_team\_guest = t.id\_team\_per\_season THEN (mr.goal\_guest\_over + mr.goal\_home\_against)

ELSE 0

END) AS goal\_over,

-- Γκολ Κατά

SUM(CASE

WHEN mr.id\_team\_home = t.id\_team\_per\_season THEN (mr.goal\_guest\_over + mr.goal\_home\_against)

WHEN mr.id\_team\_guest = t.id\_team\_per\_season THEN (mr.goal\_home\_over + mr.goal\_guest\_against)

ELSE 0

END) AS goal\_against

FROM TEAM\_PER\_SEASON AS t

LEFT JOIN MATCH\_RESULT AS mr ON t.id\_team\_per\_season IN (mr.id\_team\_home, mr.id\_team\_guest)

GROUP BY t.id\_team\_per\_season

),

Penalties AS (

SELECT

tp.id\_team\_per\_season,

SUM(p.points\_penalty) AS points\_penalty

FROM TEAM\_PUNISHMENT AS tp

JOIN PUNISHMENT AS p ON tp.id\_punishment = p.id\_punishment

GROUP BY tp.id\_team\_per\_season

),

TeamInfo AS (

SELECT

t.id\_team\_per\_season AS id,

cl.name AS name,

l.name AS league1,

s.year

FROM TEAM\_PER\_SEASON AS t

JOIN CLUB AS cl ON t.EPO\_id = cl.EPO\_id

JOIN LEAGUE AS l ON t.id\_league = l.id\_league

JOIN SEASON AS s ON t.id\_season = s.id\_season

)

SELECT

ti.year,

ti.league1,

ti.id,

ti.name,

ms.matches,

(3 \* ms.win + ms.draw - COALESCE(p.points\_penalty, 0)) AS points,

ms.win,

ms.defeat,

ms.draw,

ms.goal\_over,

ms.goal\_against

FROM TeamInfo AS ti

LEFT JOIN MatchStats AS ms ON ti.id = ms.id

LEFT JOIN Penalties AS p ON ti.id = p.id\_team\_per\_season

ORDER BY ti.year, ti.league1, points DESC;

* + Θέλουμε για κάθε σεζόν ανά κατηγορία τους 10 πρώτους σκόρερς, δηλαδή το id, το όνομα και το επίθετο κάθε παίκτη, τα συνολικά γκολ (όχι αυτογκόλ) που έχει σκοράρει και την ομάδα στην οποία αγωνίζεται. Οι παίκτες πρέπει να εμφανίζονται σε φθίνουσα σειρά τερμάτων. Ο κώδικας που αναπτύχθηκε είναι ο παρακάτω:

WITH RankedPlayers AS (

SELECT

s.year,

l.name AS league\_name,

p.id\_player,

p.name AS player\_name,

p.lastname,

COUNT(pg.id\_goal) AS goals\_over,

cl.name AS club\_name,

ROW\_NUMBER() OVER (PARTITION BY l.name ORDER BY COUNT(pg.id\_goal) DESC ,p.lastname) AS rank

FROM PLAYER AS p

JOIN PLAYER\_GOAL AS pg ON p.id\_player = pg.id\_player

JOIN GOAL AS g ON pg.id\_goal = g.id\_goal

JOIN TEAM\_PLAYER AS tp ON tp.id\_player = p.id\_player

JOIN TEAM\_PER\_SEASON AS t ON t.id\_team\_per\_season = tp.id\_team\_per\_season

JOIN CLUB AS cl ON cl.EPO\_id = t.EPO\_id

JOIN LEAGUE AS l ON l.id\_league = t.id\_league

JOIN SEASON AS s ON s.id\_season = t.id\_season

WHERE g.type = "γκολ\_υπέρ"

GROUP BY s.year, l.name, p.id\_player, p.name, p.lastname, cl.name)

SELECT

year,

league\_name,

id\_player,

player\_name,

lastname,

goals\_over,

club\_name

FROM RankedPlayers

WHERE rank <= 10

ORDER BY league\_name, rank;

* + Θέλουμε για κάθε σεζόν ανά κατηγορία τους παίκτες με τις περισσότερες κίτρινες κάρτες, δηλαδή το id, το όνομα και το επίθετο κάθε παίκτη, τις συνολικές κίτρινες κάρτες που έχει δεχτεί και την ομάδα στην οποία αγωνίζεται. Οι παίκτες πρέπει να εμφανίζονται σε φθίνουσα σειρά κίτρινων καρτών. Ο κώδικας που αναπτύχθηκε είναι ο παρακάτω:

SELECT s.year, l.name, p.id\_player, p.name, p.lastname, count(pc.id\_card) as yellow\_cards, cl.name as club\_name

FROM(((((((PLAYER as p JOIN PLAYER\_CARD as pc on p.id\_player = pc.id\_player) JOIN CARD as c on pc.id\_card = c.id\_card) JOIN TEAM\_PLAYER as tp on tp.id\_player = p.id\_player)JOIN TEAM\_PER\_SEASON as t on t.id\_team\_per\_season = tp.id\_team\_per\_season)JOIN CLUB as cl on cl.EPO\_id = t.EPO\_id)JOIN LEAGUE as l on l.id\_league = t.id\_league)join season as s on s.id\_season = t.id\_season)

WHERE c.type = "κίτρινη"

GROUP BY p.id\_player

ORDER BY l.id\_league, yellow\_cards desc , p.lastname;

* + Θέλουμε για κάθε σεζόν ανά κατηγορία τους παίκτες με τις περισσότερες κόκκινες κάρτες, δηλαδή το id, το όνομα και το επίθετο κάθε παίκτη, τις συνολικές κόκκινες κάρτες που έχει δεχτεί και την ομάδα στην οποία αγωνίζεται. Οι παίκτες πρέπει να εμφανίζονται σε φθίνουσα σειρά κόκκινων καρτών. Ο κώδικας που αναπτύχθηκε είναι ο παρακάτω:

SELECT s.year, l.name, p.id\_player, p.name, p.lastname, count(pc.id\_card) as red\_cards, cl.name as club\_name

FROM(((((((PLAYER as p JOIN PLAYER\_CARD as pc on p.id\_player = pc.id\_player) JOIN CARD as c on pc.id\_card = c.id\_card) JOIN TEAM\_PLAYER as tp on tp.id\_player = p.id\_player)JOIN TEAM\_PER\_SEASON as t on t.id\_team\_per\_season = tp.id\_team\_per\_season)JOIN CLUB as cl on cl.EPO\_id = t.EPO\_id)JOIN LEAGUE as l on l.id\_league = t.id\_league)join season as s on s.id\_season = t.id\_season)

WHERE c.type = "κόκκινη"

GROUP BY p.id\_player

ORDER BY l.id\_league, red\_cards desc, p.lastname;

* + Θέλουμε για κάθε σεζόν ανά κατηγορία τους παίκτες με τα περισσότερα λεπτά συμμετοχής, δηλαδή το id, το όνομα και το επίθετο κάθε παίκτη, τα λεπτά συμμετοχής, το πλήθος των αγώνων στους οποίους έχει λάβει μέρος και την ομάδα στην οποία αγωνίζεται. Οι παίκτες πρέπει να εμφανίζονται σε φθίνουσα σειρά λεπτών συμμετοχής. Ο κώδικας που αναπτύχθηκε είναι ο παρακάτω:

SELECT s.year, l.name, cl.name as club\_name, p.id\_player, p.name, p.lastname, sum(pa.minute\_out - pa.minute\_in) as participation\_minutes, count(pp.id\_match) as matches

FROM(((((((PLAYER as p JOIN PLAYER\_PARTICIPATION as pp on p.id\_player = pp.id\_player) JOIN PARTICIPATION as pa on pp.id\_participation = pa.id\_participation) JOIN TEAM\_PLAYER as tp on tp.id\_player = p.id\_player)JOIN TEAM\_PER\_SEASON as t on t.id\_team\_per\_season = tp.id\_team\_per\_season)JOIN CLUB as cl on cl.EPO\_id = t.EPO\_id)JOIN LEAGUE as l on l.id\_league = t.id\_league)join season as s on s.id\_season = t.id\_season)

GROUP BY p.id\_player

ORDER BY l.id\_league, participation\_minutes desc, p.lastname ASC;

* + Θέλουμε κάθε σεζόν ανά κατηγορία για κάθε παίκτη κάθε ομάδας τους αγώνες στους οποίους έχει συμμετάσχει, δηλαδή το ονοματεπώνυμο του παίκτη, το id και την ημερομηνία του κάθε αγώνα, τα λεπτά συμμετοχής, τα γκολ υπέρ και τα αυτογκόλ που σκόραρε, τις κίτρινες και κόκκινες κάρτες που δέχτηκε στο συγκεκριμένο ματς, καθώς και το αν συμμετείχε στον αγώνα ως αλλαγή ή ως βασικός (αν το minute\_in είναι 0 είναι βασικός, αλλιώς μπαίνει ως αλλαγή). Στο σημείο, αυτό, αποφασίσαμε το συγκεκριμένο ερώτημα να μετατραπεί σε View, ώστε τα αποτελέσματά του να μπορούμε δυνητικά να τα αξιοποιήσουμε και σε επόμενα ερωτήματα και να μην χρειαστεί να τα υπολογίσουμε ξανά. Ο κώδικας, λοιπόν, που αναπτύχθηκε είναι ο ακόλουθος:

CREATE VIEW MATCHES\_PLAYERS AS

SELECT

s.year,

l.name AS league\_name,

m.id\_match,

m.match\_date,

cl.name AS club\_name,

t.id\_team\_per\_season,

p.id\_player,

p.name AS player\_name,

p.lastname AS player\_lastname,

((mr.goal\_home\_over + mr.goal\_guest\_against) || ' - ' || (mr.goal\_guest\_over + mr.goal\_home\_against)) AS score,

(pa.minute\_out - pa.minute\_in) AS minutes\_of\_participation,

SUM(CASE WHEN pg.id\_goal = 1 THEN 1 ELSE 0 END) AS goals\_over,

SUM(CASE WHEN pg.id\_goal = 2 THEN 1 ELSE 0 END) AS goals\_against,

SUM(CASE WHEN pc.id\_card = 1 THEN 1 ELSE 0 END) AS yellow\_cards,

SUM(CASE WHEN pc.id\_card = 2 THEN 1 ELSE 0 END) AS red\_cards,

CASE

WHEN pa.minute\_in = 0 THEN 'βασικός'

ELSE 'αλλαγή'

END AS player\_status

FROM

MATCH AS m

JOIN MATCHWEEK AS mw ON m.id\_matchweek = mw.id\_matchweek

JOIN SEASON AS s ON mw.id\_season = s.id\_season

JOIN LEAGUE AS l ON l.id\_league = mw.id\_league

JOIN TEAM\_PER\_SEASON AS t ON (t.id\_team\_per\_season = m.id\_team\_home OR t.id\_team\_per\_season = m.id\_team\_guest)

JOIN CLUB AS cl ON cl.EPO\_id = t.EPO\_id

JOIN TEAM\_PLAYER AS tp ON tp.id\_team\_per\_season = t.id\_team\_per\_season

JOIN PLAYER AS p ON p.id\_player = tp.id\_player

LEFT JOIN PLAYER\_GOAL AS pg ON pg.id\_player = p.id\_player AND pg.id\_match = m.id\_match

LEFT JOIN PLAYER\_CARD AS pc ON pc.id\_player = p.id\_player AND pc.id\_match = m.id\_match

JOIN MATCH\_RESULT AS mr ON m.id\_match = mr.id\_match

JOIN PLAYER\_PARTICIPATION AS pp ON pp.id\_player = p.id\_player AND pp.id\_match = m.id\_match

JOIN PARTICIPATION AS pa ON pp.id\_participation = pa.id\_participation

GROUP BY

s.year,

l.name,

m.match\_date,

m.id\_match,

cl.name,

t.id\_team\_per\_season,

p.id\_player

ORDER BY

p.id\_player--s.year, l.name, m.match\_date, m.id\_match, cl.name, p.lastname;

* + Θέλουμε για κάθε ποδοσφαιριστή στατιστικά για κάθε σεζόν. Πιο συγκεκριμένα, θέλουμε το id και το ονοματεπώνυμο του παίκτη, τα συνολικά γκολ και αυτογκόλ που έχει σκοράρει, τις συνολικές κάρτες, κόκκινες και κίτρινες, που έχει δεχτεί, τα συνολικά λεπτά συμμετοχής και τους συνολικούς αγώνες στους οποίους έχει συμμετάσχει σε μια σεζόν. Για να τα υπολογίσουμε όλα αυτά αξιοποιήσαμε το view που δημιουργήσαμε στο προηγούμενο ερώτημα. Οπότε, ο πίνακας που αναπτύχθηκε είναι ο παρακάτω:

SELECT mp.year, mp.id\_player, mp.player\_name, mp.player\_lastname, sum(mp.goals\_over), sum(mp.goals\_against), sum(mp.yellow\_cards), sum(mp.red\_cards), sum(mp.minutes\_of\_participation), count(mp.id\_match)

FROM MATCHES\_PLAYERS as mp

GROUP BY mp.year, mp.id\_player;

* + Θέλουμε να καταγράφουμε τις ομάδες που πέφτουν κατηγορία, δηλαδή το όνομα των ομάδων, την κατηγορία από την οποία έπεσαν και την κατηγορία στην οποία βρέθηκαν. Οι ομάδες που πέφτουν κατηγορία είναι αυτές που βρίσκονται χαμηλότερα (η τελευταία ομάδα κάθε κατηγορίας) στην κατάταξη με βάση τη βαθμολογία τους. Αν περισσότερες από μία ομάδες έχουν την ίδια βαθμολογία, υποβιβάζεται η ομάδα με το μικρότερο goal difference (γκολ\_υπέρ – γκολ\_κατά), όπως συμβαίνει και σε μια πραγματική ΕΠΣ. Τέλος, υποβιβασμό μπορούν να υποστούν μόνο ομάδες που ανήκουν στην Α και στη Β κατηγορία. Οπότε, ο κώδικας που αναπτύχθηκε είναι ο ακόλουθος:

WITH RankedTeams AS (

SELECT id, league1, name, ROW\_NUMBER() OVER (PARTITION BY league1 ORDER BY points ASC, (goal\_over - goal\_against) ASC, goal\_over DESC) AS rank

FROM RANKING

)

SELECT r.league1, GROUP\_CONCAT(' (' || r.name || ' , ' || r.id || ')') AS relegated\_teams, (r.league1 || ' -> ' || COALESCE(l\_next.name, ''))AS leagues

FROM RankedTeams AS r

JOIN LEAGUE AS l1 ON r.league1 = l1.name

LEFT JOIN LEAGUE AS l\_next ON l\_next.id\_league = l1.id\_league + 1

WHERE r.rank = 1 AND r.league1 <> 'Γ'

GROUP BY r.league1, l1.id\_league;

* + Θέλουμε να καταγράφουμε τις ομάδες που ανεβαίνουν κατηγορία, δηλαδή το όνομα των ομάδων, την κατηγορία από την οποία ανέβηκαν και την κατηγορία στην οποία βρέθηκαν. Οι ομάδες που ανεβαίνουν κατηγορία είναι αυτές που βρίσκονται ψηλότερα ( η πρώτη ομάδα κάθε κατηγορίας) στην κατάταξη με βάση τη βαθμολογία τους. Αν περισσότερες από μία ομάδες έχουν την ίδια βαθμολογία, ανεβαίνει η ομάδα με το μεγαλύτερο goal difference (γκολ\_υπέρ – γκολ\_κατά), όπως συμβαίνει και σε μια πραγματική ΕΠΣ. Τέλος, άνοδο μπορούν να πετύχουν μόνο ομάδες που ανήκουν στην Β και στη Γ κατηγορία. Οπότε, ο κώδικας που αναπτύχθηκε είναι ο ακόλουθος:

WITH RankedTeams AS (

SELECT id, league1, name, ROW\_NUMBER() OVER (PARTITION BY league1 ORDER BY points DESC, (goal\_over - goal\_against) DESC, goal\_over DESC) AS rank

FROM RANKING

)

SELECT r.league1, GROUP\_CONCAT(' (' || r.name || ' , ' || r.id || ')') AS promoted\_teams, (r.league1 || ' -> ' || COALESCE(l\_next.name, ''))AS leagues

FROM RankedTeams AS r

JOIN LEAGUE AS l1 ON r.league1 = l1.name

LEFT JOIN LEAGUE AS l\_next ON l\_next.id\_league = l1.id\_league - 1

WHERE r.rank = 1 AND r.league1 <> 'Α'

GROUP BY r.league1, l1.id\_league;

* + Επειδή θέλαμε να αξιοποιήσουμε και την πράξη της διαίρεσης, αποφασίσαμε να βρούμε ανά κατηγορία τις ομάδες που έχουν σκοράρει τουλάχιστον 1 γκολ σε όλα τα εντός έδρας ματς. Ο κώδικας που αναπτύχθηκε είναι ο ακόλουθος:

SELECT DISTINCT l.name AS league\_name, cl.name AS club\_name

FROM (TEAM\_PER\_SEASON AS t JOIN CLUB AS cl ON t.EPO\_id = cl.EPO\_id JOIN LEAGUE AS l ON t.id\_league = l.id\_league)

WHERE NOT EXISTS (

SELECT 1

FROM MATCH AS m

WHERE (m.id\_team\_home = t.id\_team\_per\_season) --OR m.id\_team\_guest = t.id\_team\_per\_season)

AND NOT EXISTS (

SELECT 1

FROM MATCH\_RESULT AS mr

WHERE mr.id\_match = m.id\_match

AND (

(mr.id\_team\_home = t.id\_team\_per\_season AND mr.goal\_home\_over > 0)

/\*OR

(mr.id\_team\_guest = t.id\_team\_per\_season AND mr.goal\_guest\_over > 0)\*/

)

)

)

AND EXISTS

SELECT 1

FROM MATCH AS m2

WHERE m2.id\_team\_home = t.id\_team\_per\_season OR m2.id\_team\_guest = t.id\_team\_per\_season

);

* + Θέλουμε ανά κατηγορία να βρούμε τους παίκτες που έχουν βάλει γκολ από αλλαγή. Δηλαδή θέλουμε το id, το όνομα και το επίθετο του παίκτη που σκόραρε από αλλαγή, καθώς και το id του ματς στο οποίο αυτό συνέβη. Ο κώδικας που αναπτύχθηκε είναι ο παρακάτω:

SELECT mp.year, mp.id\_match, mp.league\_name, mp.id\_player, mp.player\_name, mp.player\_lastname

FROM MATCHES\_PLAYERS AS mp

WHERE mp.player\_status = 'αλλαγή' AND mp.goals\_over >= 1 --AND mp.score <> '0 - 0'

GROUP BY mp.id\_match;

* + Θέλουμε ανά κατηγορία για τις ομάδες που έχουν κερδίσει σε πάνω από τους μισούς αγώνες στους οποίους έχουν συμμετάσχει, το ποσοστό από αυτές τις νίκες που επετεύχθησαν σε εντός έδρας αγώνες. Είναι ένα χρήσιμο ερώτημα για να παρατηρήσουμε αν οι ομάδες που έχουν αρκετές νίκες είναι πιο καλές στην έδρα τους. Ο κώδικας που αναπτύχθηκε είναι ο παρακάτω:

SELECT r.league1, r.id, r.name,((r.win \* 100)/r.matches) as percentage\_win,((select count(mr.id\_match)

FROM MATCH\_RESULT as mr

WHERE (mr.id\_team\_home = r.id and (mr.goal\_home\_over + mr.goal\_guest\_against > mr.goal\_guest\_over + mr.goal\_home\_against))) \*100/r.win) as percentage\_home\_win

FROM RANKING as r

GROUP BY r.id

HAVING ((r.win \* 100)/r.matches)>=50;

1. ΜΕΘΟΔΟΛΟΓΙΑ – ΔΗΜΙΟΥΡΓΙΑ Indexes

Μετά τη σύνταξη των queries αποφασίσαμε να δημιουργήσουμε μερικά χρήσιμα indexes, ώστε να επιταχυνθεί η αναζήτηση των δεδομένων που προκύπτει ως ανάγκη από την υποβολή των προαναφερθέντων ερωτημάτων. Η δημιουργία των indexes,λοιπόν, έγινε με κριτήριο τις συνενώσεις πινάκων και τις συνθήκες where που θέταμε σε καθένα από τα ερωτήματα, το feedback που λαμβάναμε από το “explain query plan” για κάθε select, αλλά και με γνώμονα ποια ερωτήματα μας ενδιαφέρει να επιταχύνουμε σε μεγαλύτερο βαθμό. Ταυτόχρονα, φροντίσαμε να υλοποιήσουμε και indexes που περιλαμβάνουν γνωρίσματα που λαμβάνουν μέρος στη δημιουργία των Views, ώστε να επιταχυνθούν σχετικά με αυτά τα Views ερωτήματα.

Παρατηρήσαμε, λοιπόν, ότι η δημιουργία των indexes επιτάχυνε σε μεγάλο βαθμό ερωτήματα, σε κάποια δε ο χρόνος εκτέλεσης μειώθηκε στο 1/3. Κάποια παραδείγματα χρόνων πριν και μετά την υλοποίηση των Indexes φαίνονται παρακάτω.

* + Για το ερώτημα με το View “MATCHES\_PLAYERS”:

| Χρόνος εκτέλεσης πριν τη δημιουργία του index | Χρόνος εκτέλεσης μετά τη δημιουργία του index |
| --- | --- |
|  |  |

* + Για το ερώτημα με τα στατιστικά των παικτών:

| Χρόνος εκτέλεσης πριν τη δημιουργία του index | | Χρόνος εκτέλεσης μετά τη δημιουργία του index |
| --- | --- | --- |
|  |  | |

* + Για το ερώτημα με τις κίτρινες κάρτες:

| Χρόνος εκτέλεσης πριν τη δημιουργία του index | Χρόνος εκτέλεσης μετά τη δημιουργία του index |
| --- | --- |
|  |  |

1. Δημιουργια εφαρμογησ – γραφικο περιβαλλον

Μετά και από τη δημιουργία των indexes, η βάση μας είχε στηθεί, οπότε έπρεπε να δημιουργήσουμε ένα γραφικό περιβάλλον, ώστε να μπορούν οι χρήστες να έχουν πρόσβαση στα δεδομένα της βάσης μας και να ενημερώνονται για τα θέματα του τοπικού πρωταθλήματος μέσω των αποτελεσμάτων των ερωτημάτων που αποφασίσαμε να θέσουμε σε αυτή.

Σε πρώτη φάση έπρεπε να αποφασίσουμε με ποιον ρόλο θα μπορεί κάθε χρήστης να αξιοποιήσει την εφαρμογή μας και ποιες δυνατότητες θα έχει ανάλογα με τον ρόλο που διαδραματίζει. Σκεφτήκαμε, λοιπόν, οι ρόλοι να είναι τρεις, αυτός του απλού “user”, αυτός του “roster\_admin”, κι αυτός του “admin”. Η ιεραρχία σχετικά με τις ενέργειες στις οποίες δύναται να προβεί κάθε χρήστης στην εφαρμογή μας είναι σύμφωνη με τη σειρά που αναφέρθηκαν οι ρόλοι (από τα λιγότερα στα περισσότερα προνόμια). Εδώ αξίζει να σημειωθεί ότι τις δυνατότητες που έχουν οι χρήστες που ανήκουν στην προηγούμενη βαθμίδα τις έχουν και εκείνοι που ανήκουν στην επόμενη συν κάποιες ακόμα. Ας αναφερθούμε, όμως, σε κάθε ρόλο με τη σειρά παραθέτοντας εκτός από τις δυνατότητές του και το γραφικό περιβάλλον με το οποίο έρχεται σε επαφή.

* 1. Ο ρόλος του user

Τον ρόλο του απλού user διαθέτει οποιοσδήποτε χρήστης ανοίγει την εφαρμογή μας και έχει πρόσβαση σε συγκεκριμένες λειτουργίες. Οι λειτουργίες αυτές αφορούν στοιχεία για τις ομάδες και τους παίκτες.

Πιο συγκεκριμένα, ο user μπορεί να ενημερωθεί για θέματα που αφορούν τις **ομάδες**:

* + Χαρακτηριστικά ομάδων διαχωρισμένες στις κατηγορίες που ανήκουν

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, ιστοσελίδα

Περιγραφή που δημιουργήθηκε αυτόματα

* + Πρόγραμμα αγώνων των ομάδων διαχωρισμένες στις κατηγορίες που ανήκουν

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, ιστοσελίδα

Περιγραφή που δημιουργήθηκε αυτόματα

* + Βαθμολογία των ομάδων στην κατηγορία τους

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, εικονίδιο υπολογιστή

Περιγραφή που δημιουργήθηκε αυτόματα

* + Ρόστερ των ομάδων. Εδώ υπάρχει δυνατότητα επιλογής της ομάδας. Επιπλέον, αν πατήσει ο χρήστης πάνω σε κάποιον παίκτη έχει τη δυνατότητα να δει τους συνολικούς αγώνες στους οποίους έχει αγωνιστεί μέσα στη σεζόν, καθώς και τα συνολικά στατιστικά του

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, ιστοσελίδα

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, ιστοσελίδα

Περιγραφή που δημιουργήθηκε αυτόματα

Επιλογή ποδοσφαιριστή

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, εικονίδιο υπολογιστή

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα που περιέχει κείμενο, λογισμικό, λειτουργικό σύστημα, εικονίδιο υπολογιστή

Περιγραφή που δημιουργήθηκε αυτόματα

* + Ποινές που έχουν λάβει ομάδες διάφορων κατηγοριών

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, εικονίδιο υπολογιστή

Περιγραφή που δημιουργήθηκε αυτόματα

* + Ομάδες που έπεσαν κατηγορία

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, λειτουργικό σύστημα

Περιγραφή που δημιουργήθηκε αυτόματα

* + Ομάδες που ανέβηκαν κατηγορία

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, λειτουργικό σύστημα

Περιγραφή που δημιουργήθηκε αυτόματα

Επιπλέον, ο user μπορεί να ενημερωθεί για top 10 παικτών σε διάφορες κατηγορίες:

* Top 10 scorers

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, ιστοσελίδα

Περιγραφή που δημιουργήθηκε αυτόματα

* Top 10 παικτών σε κίτρινες κάρτες

Εικόνα που περιέχει κείμενο, λογισμικό, ιστοσελίδα, λειτουργικό σύστημα

Περιγραφή που δημιουργήθηκε αυτόματα

* Top 10 παικτών σε κόκκινες κάρτες

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, ιστοσελίδα

Περιγραφή που δημιουργήθηκε αυτόματα

* Top 10 παικτών σε λεπτά συμμετοχής

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, ιστοσελίδα

Περιγραφή που δημιουργήθηκε αυτόματα

Τέλος, ο user μπορεί να ενημερωθεί για κάποια fun facts που αφορούν παίκτες ή ομάδες και αφορούν:

* Ομάδες που έχουν σκοράρει τουλάχιστον ένα γκολ σε όλα τα εντός έδρα παιχνίδια τους

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, λειτουργικό σύστημα

Περιγραφή που δημιουργήθηκε αυτόματα

* Παίκτες που έχουν σκοράρει γκολ από αλλαγή

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, ιστοσελίδα

Περιγραφή που δημιουργήθηκε αυτόματα

* Πόσες (σε ποσοστό) είναι οι νίκες σε εντός έδρας ματς για ομάδες που έχουν νικήσει σε πάνω από τα μισά παιχνίδια στα οποία έχουν συμμετάσχει

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, λειτουργικό σύστημα

Περιγραφή που δημιουργήθηκε αυτόματα

* 1. Ο ρόλος του roster\_admin

Ο ρόλος του roster\_admin απευθύνεται για παράδειγμα σε διαχειριστές των σωματείων που αγωνίζονται στο τοπικό. Με τον ρόλο αυτό κάποιος έχει πρόσβαση σε όσα έχει κι ένα απλός user, αλλά επιπλέον μπορεί να κάνει Login με το Username και το Password του στην εφαρμογή μας και να προσθέσει κάποιον παίκτη σε ρόστερ ομάδας κάποιας σεζόν, αλλά και να αλλάξει το password του. Οι προαναφερθείσες λειτουργίες φαίνονται παρακάτω:

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, διάγραμμα

Περιγραφή που δημιουργήθηκε αυτόματα Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

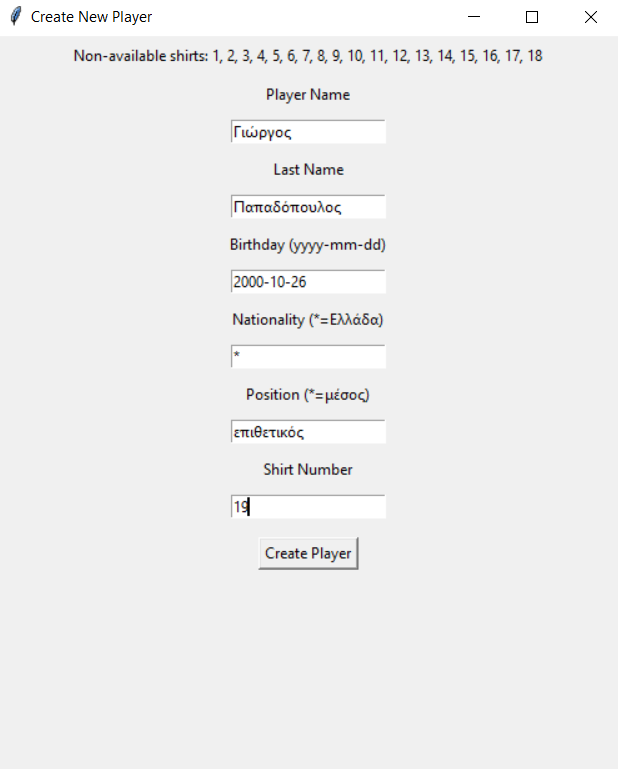
Περιγραφή που δημιουργήθηκε αυτόματα

Παράθυρο Login Παράθυρο επιλογών (εδώ Εισαγωγή παίκτη)

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό

Περιγραφή που δημιουργήθηκε αυτόματα

Επιλογή ομάδας όπου θα ενταχθεί ο παίκτης



Εισαγωγή των στοιχείων του παίκτη

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Μήνυμα επιτυχούς αποθήκευσης του παίκτη Μήνυμα επιτυχούς εισαγωγής του παίκτη σε ομάδα

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, εικονίδιο υπολογιστή

Περιγραφή που δημιουργήθηκε αυτόματα Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Επιλογή αλλαγής password Αλλαγή password

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, γραμμή

Περιγραφή που δημιουργήθηκε αυτόματα

Μήνυμα επιτυχούς καταχώρησης νέου password Μήνυμα επιτυχούς αποσύνδεσης

* 1. Ο ρόλος του admin

Ο ρόλος του admin απευθύνεται στον διαχειριστή της εφαρμογής. Ο admin, λοιπόν, σε αυτό το στάδιο που βρίσκεται η εφαρμογή μας (σίγουρα επιδέχεται βελτιώσεις) έχει τις ίδιες δυνατότητες με έναν roster\_admin, ενώ επιπλέον μπορεί να δημιουργήσει ή να διαγράψει έναν roster\_admin. Οι επιπρόσθετες, αυτές, λειτουργίες φαίνονται παρακάτω:

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Επιλογές του admin

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Δημιουργία του roster\_admin Μήνυμα επιτυχούς δημιουργίας του roster\_admin

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Διαγραφή roster\_admin “beta” Μήνυμα επιτυχούς διαγραφής του “beta”

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, λογότυπο

Περιγραφή που δημιουργήθηκε αυτόματα

Μήνυμα σφάλματος σύνδεσης

1. ΔΙΑΜΟΙΡΑΣΜΟΣ ΕΡΓΑΣΙΩΝ

Το παρόν πρότζεκτ αποτελείται από πολλά σκέλη, με αποτέλεσμα να έχει αρκετό φόρτο εργασίας και να απαιτεί στενή και καλή συνεργασία για τη δημιουργία και τη σωστή διασύνδεσή τους. Συνεπώς, δουλέψαμε κάποια κομμάτια συνεργατικά δια ζώσης και άλλα κομμάτια εξ αποστάσεως αξιοποιώντας εργαλεία όπως το AnyDesk και το github, ώστε να μπορούμε να μοιράσουμε δίκαια τις εργασίες και να συντονιστούμε σωστά για την ομαλή ολοκλήρωση της εργασίας.

Αν και απαιτείται, λοιπόν, στενή συνεργασία και ενασχόληση και των δύο σε όλα τα μέρη της εργασίας, ο καθένας επικεντρώθηκε σε περισσότερο σε συγκεκριμένα σκέλη. Ο διαμοιρασμός που ακολουθήσαμε είναι ο εξής:

* Δημιουργία ERD: από κοινού (διά ζώσης)
* Δημιουργία Schema: από κοινού (διά ζώσης)
* Δημιουργία βάσης και διασφάλιση αναφορικής ακεραιότητας: από κοινού (διά ζώσης)
* Εισαγωγή δεδομένων στη βάση με ai: Α. Καγιάφα
* Εισαγωγή δεδομένων στη βάση με python: Α. Μενδρινός
* Επιλογή SQL queries: από κοινού
* Διατύπωση SQL queries: Α. Καγιάφα
* CRUD queries: Α. Μενδρινός
* Αξιολόγηση λειτουργιών CRUD: από κοινού
* Δημιουργία Indexes: Α. Μενδρινός
* Δημιουργία GUI εφαρμογής: Α. Καγιάφα

1. ΧΡΟΝΟΔΙΑΓΡΑΜΜΑ

Το χρονοδιάγραμμα που ακολουθήθηκε, ώστε να ολοκληρώσουμε την παρούσα εργασία είναι το παρακάτω:

* Δημιουργία ERD: Τέλη Οκτωβρίου
* Δημιουργία Schema: Τέλη Οκτωβρίου
* Δημιουργία βάσης και διασφάλιση αναφορικής ακεραιότητας: Αρχές Δεκεμβρίου
* Εισαγωγή δεδομένων στη βάση με ai: Αρχές Δεκεμβρίου
* Εισαγωγή δεδομένων στη βάση με python: Αρχές Δεκεμβρίου
* Επιλογή SQL queries: Αρχές Δεκεμβρίου
* Διατύπωση SQL queries: Διακοπές Χριστουγέννων
* CRUD queries: Διακοπές Χριστουγέννων
* Αξιολόγηση λειτουργιών CRUD: Διακοπές Χριστουγέννων
* Δημιουργία Indexes: Διακοπές Χριστουγέννων
* Δημιουργία GUI εφαρμογής: Διακοπές Χριστουγέννων

1. αξιολογηση