ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Δημιουργία εφαρμογής για τη μετατροπή μίας Αντικειμενοστρεφούς Βάσης Δεδομένων σε Linked Data

Του φοιτητή

Ζαζόπουλος Ευκλείδης

Αρ. Μητρώου: 01/1666

Επιβλέπων καθηγητής

Κεραμόπουλος Ευκλείδης

Θεσσαλονίκη 2014

ΠΡΟΛΟΓΟΣ

Ένας νέος διαδικτυακός κόσμος γεννιέται. Ένας κόσμος όπου τα πάντα ανήκουν κάπου και συνδέονται με κάτι άλλο. Κάθε δεδομένο οδηγεί σε κάποιο άλλο και το μόνο που έχουμε να κάνουμε είναι να ακολουθήσουμε την διαδρομή. Ο Σημασιολογικός Ιστός (Semantic Web) και τα Συνδεδεμένα δεδομένα (Linked Data) είναι γεγονός.

Η παρούσα πτυχιακή στοχεύει στο να βρει κάποιον σύνδεσμο μεταξύ των ευρέως χρησιμοποιούμενων Αντικειμενοστρεφών Βάσεων Δεδομένων και των Συνδεδεμένων Δεδομένων ώστε η μετάβαση να έρθει ένα βήμα πιο κοντά.

Η υλοποίηση αυτής της εφαρμογής έγινε με τη βοήθεια της αντικειμενοστρεφούς γλώσσας προγραμματισμού Java και τις βιβλιοθήκες γραφικών (GUI widget toolkit) Java Swing. Για τις ανάγκες της Αντικειμενοστρεφούς Βάσης Δεδομένων χρησιμοποιήθηκε η ObjectDB με τη βοήθεια του Java Persistence API. Τέλος τα αποτελέσματα της επεξεργασίας των δεδομένων εκφράζονται σε RDF Turtle μορφή.

Στα κεφάλαια που ακολουθούν περιγράφονται αναλυτικά οι τεχνολογίες που χρησιμοποιήθηκαν μαζί με αποσπάσματα κώδικα και τις αντίστοιχες επεξηγήσεις τους και παρατίθεται ένας οδηγός χρήσης της εφαρμογής

ABSTRACT

A new internet world is being formed. A world where everything belongs somewhere and connects to something else. Every single data leads to another and all we have to do is enjoy the ride. Semantic Web and Linked Data is a fact.

This graduation projects aims at finding a link between Object Oriented Databases and Linked Data to make the transition easier.

The implementation of this application was done with the support of the object oriented programming language Java and it's GUI widget toolkit Java Swing. For the needs of the object oriented database, ObjectDB was used with the support of Java Persistence API. At last the results where introduced in a RDF Turlte form.

The following chapters detail the technologies that were used along with fragments of code explained and is cited a user guide for the created application.

ΕΥΧΑΡΙΣΤΙΕΣ (προαιρετικά)

Θα ήθελα να ευχαριστήσω πρώτα από όλους τον Δρ. Κεραμόπουλο Ευκλείδη που ήταν υπομονετικός μαζί μου και απαντούσε στα επίμονα email μου όταν οι περισσότεροι βρίσκονταν στις παραλίες και γιατί μου έδωσε την ευκαιρία να ασχοληθώ και να μελετήσω ένα θέμα το οποίο αναμένεται να κυριαρχήσει στο Διαδίκτυο του μέλλοντος.

Επίσης ένα μεγάλο ευχαριστώ αξίζουν οι καθηγητές του τμήματος Πληροφορικής του ΑΤΕΙ Θεσσαλονίκης για την αφοσίωση και τον ζήλο που επιδεικνύουν στο λειτούργημα τους

Τέλος θέλω να ευχαριστήσω την οικογένεια μου που με στήριξε όπως κανείς άλλος όλα αυτά τα χρόνια.

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΡΟΛΟΓΟΣ 2
ABSTRACT 3
ΕΥΧΑΡΙΣΤΙΕΣ 4
ΠΕΡΙΕΧΟΜΕΝΑ 5
ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ 6
ΕΙΣΑΓΩΓΗ 7
ΚΕΦΑΛΑΙΟ 1 – Δημιουργία Αντικειμενοστρεφούς Βάσης Δεδομένων OlympicGames 21
ΕΙΣΑΓΩΓΗ 21
1.1 Δημιουργία Project στο NetBeans 22
1.2 Δημιουργία Serializable κλάσεων 23
1.3 Persistence.xml 25
1.4 Εισαγωγή Αντικειμένων στην Βάση Δεδομένων 26
Επίλογος 28
ΚΕΦΑΛΑΙΟ 2 – Ανάκτηση Δεδομένων από την ObjectDB 28
ΕΙΣΑΓΩΓΗ 28
2.1 Peristence Context 29
2.2 Τρόποι Ανάκτησης Αντικειμένων από την ObjectDB 30
2.3 Interface Metamodel 33
2.4 Συλλογή αποτελεσμάτων 34
Επίλογος 35
ΚΕΦΑΛΑΙΟ 3 – Μετατροπή αποτελεσμάτων σε RDF Turtle έγγραφο 35
ΕΙΣΑΓΩΓΗ 35
3.1 RDF Turlte 36
3.2 Μορφοποίηση δεδομένων σε Turtle μορφή 37
3.3 Έλεγχος Αποτελεσμάτων 40
ΕΠΙΛΟΓΟΣ 42
ΚΕΦΑΛΑΙΟ 4 – Οδηγός χρήσης της εφαρμογής 42

ΣΥΜΠΕΡΑΣΜΑΤΑ 47
ΒΙΒΛΙΟΓΡΑΦΙΑ.....51
ΠΑΡΑΡΤΗΜΑΤΑ.....52

Ευρετήριο σχημάτων

Σχήμα 1..... 14
Σχήμα 2..... 16
Σχήμα 3..... 18
Σχήμα 4..... 19
Σχήμα 5..... 22
Σχήμα 6..... 23
Σχήμα 7..... 29
Σχήμα 8..... 42
Σχήμα 9..... 43
Σχήμα 10..... 43
Σχήμα 11..... 44
Σχήμα 12..... 45
Σχήμα 13..... 45
Σχήμα 14..... 46
Σχήμα 15..... 46
Σχήμα 16..... 49

ΕΙΣΑΓΩΓΗ

Περιγραφή του Προβλήματος

Μέχρι σήμερα υπάρχουν αρκετοί οδηγοί και παραδείγματα στο διαδίκτυο αναφορικά με τους κανόνες μετατροπής των εγγραφών από απλές σχεσιακές βάσεις δεδομένων σε μορφή αποδεκτή για Διασυνδεδεμένα Δεδομένα, όπως είναι για παράδειγμα το πρότυπο RDF 1.1 Turtle στο οποίο θα αναφερθούμε στη συνέχεια. Ο σκοπός της πτυχιακής είναι να δείξει ότι κάτι τέτοιο είναι εφικτό και με αντικειμενοστρεφείς βάσεις δεδομένων και να παρουσιάσει μία εφαρμογή η οποία αυτόματα μετατρέπει μία τέτοια βάση (Olympic Games) σε ένα αρχείο Turtle έτοιμο για επεξεργασία στον Σημασιολογικό Ιστό.

Αρχικά γίνεται η μετατροπή του σχήματος της βάσης δεδομένων σε αντικείμενα μέσω της γλώσσας προγραμματισμού Java. Αφού δηλώσουμε τα αντικείμενα δημιουργούμε ένα Project στο NetBeans στο οποίο εισάγουμε τις κλάσεις αυτές και τις κάνουμε Serialize μέσω της βιβλιοθήκης της ObjectDB ώστε να είναι σε επεξεργάσιμη JPA μορφή και δηλώνουμε τις κλάσεις αυτές στο Persistence.xml. Επόμενο βήμα είναι η δημιουργία προτύπων αντικειμένων για να γίνει η εισαγωγή τους ως δεδομένα στην αντικειμενοστρεφή βάση δεδομένων(Α.Β.Δ.) μας. Για να γίνει όμως αυτό πρέπει να κάνουμε και την σύνδεση στην βάση δεδομένων με τη βοήθεια του EntityManagerFactory.

Σε αυτό το σημείο έχει ολοκληρωθεί το πρώτο βήμα. Έχουμε δημιουργήσει την αντικειμενοστρεφή βάση δεδομένων μας και έχουμε εισαγει κάποια δοκιμαστικά αντικείμενα, όπως για παράδειγμα κάποιους αθλητές, αθλήματα, γήπεδα κτλ. Επόμενο βήμα είναι να μπορέσουμε να ανακτήσουμε αυτά τα δεδομένα μέσω κώδικα. Αυτό επιτυγχάνεται χρησιμοποιώντας την μέθοδο TypedQuery<> της βιβλιοθήκης της ObjectDB. Αφού ανακτήσουμε τα δεδομένα μπορούμε να τα χρησιμοποιήσουμε με όποιον τρόπο θέλουμε. Βασική προϋπόθεση είναι να γνωρίζουμε τους ορισμούς των κλάσεων που αντιπροσωπεύουν τα αντικείμενα.

Από την στιγμή που έχουμε όλα τα αντικείμενα ως δεδομένα αρχίζει η επεξεργασία τους για να τα παρουσιάσουμε στην μορφή που ορίζει το πρότυπο RDF 1.1 Turtle. Αυτό γίνεται μέσω μιας SELECT εντολής η οποία ανακτεί όλα τα ορισμένα από τον χρήστη αντικείμενα που υπάρχουν στην Α.Β.Δ.. Πλέον πρέπει να αναγνωρίσουμε ποιας κλάσεως είναι το κάθε αντικείμενο και να το παρουσιάσουμε στην σωστή μορφή.

Τέλος αποθηκεύουμε όλα τα αποτελέσματα σε ένα αρχείο (π.χ. OlympicGames.ttl) και τα δεδομένα μας είναι έτοιμα για επεξεργασία από το Semantic Web

Ο Αντικειμενοστρεφής Προγραμματισμός

Η σχολή του αντικειμενοστρεφούς προγραμματισμού διδάσκει την ανάπτυξη λογισμικού ως ενός συνόλου από αλληλεπιδρώντα αντικείμενα. Προσφέρει ένα νέο τρόπο δημιουργίας προγραμμάτων, δίνοντας βάση στην οργάνωση του κώδικα ώστε να είναι εύκολα παραμετροποιήσιμος και επαναχρησιμοποιήσιμος. Διαφοροποιείται σε σχέδη με το διαδικαστικό μοντέλο προγραμματισμού, όπου ένα πρόγραμμα αντιμετωπίζεται σαν μία σειρά εντολών που εκτελούνται σε αλληλουχία. Βασικό χαρακτηριστικό της αντικειμενοστρεφούς προσέγγισης είναι η από κοινού αντιμετώπιση της κατάστασης και της συμπεριφοράς των αντικειμένων. Κάθε αντικείμενο είναι ένα ξεχωριστό τμήμα του προγράμματος που αλληλεπιδρά με τα άλλα τμήματα με ελεγχόμενους τρόπους.

Τα αντικείμενα ορίζονται από τις κλάσεις. Οι κλάσεις είναι ουσιαστικά μία κατηγορία αντικειμένων με την ίδια συμπεριφορά. Τα αντικείμενα είναι απλά στιγμιότυπα των κλάσεων που τους δίνουν μορφή. Εξέχουσα θέση στον αντικειμενοστρεφή προγραμματισμό έχουν οι έννοιες της κληρονομικότητας και του πολυμορφισμού. Οι δημοφιλέστερες σήμερα γλώσσες που ανήκουν σε αυτήν την σχολή είναι η Java, η C++ και η C#. Και οι τρεις αυτές γλώσσες έχουν έντονες επιρροές από την γλώσσα C.

Μία από τις βασικές αρχές του αντικειμενοστρεφούς προγραμματισμού είναι η αρχή ανοιχτότητας-κλειστότητας (σύμφωνα με τον Bertrand Meyer). Ουσιαστικά εννοεί ότι οι εφαρμογές πρέπει να είναι ανοιχτές προς την επέκταση των δυνατοτήτων τους, αλλά κλειστές σε αλλαγές στην υλοποίηση τους. Μία μεγάλη επιτυχία του αντικειμενοστρεφούς προγραμματισμού είναι ότι άνοιξε τον δρόμο για την δημιουργία τεράστιων βιβλιοθηκών, έτοιμων τμημάτων δηλαδή κώδικα με συγκεκριμένες αρμοδιότητες τα οποία μπορούν να χρησιμοποιηθούν αυτούσια.

Βάσεις Δεδομένων

Με τον όρο βάση δεδομένων εννοείται μία συλλογή από μορφοποιημένα σχετιζόμενα δεδομένα τα οποία μπορούμε να ανακτήσουμε, να τροποποιήσουμε και να εμπλουτίσουμε κατ' απαίτηση. Τα προγράμματα που διαχειρίζονται τις βάσεις δεδομένων ονομάζονται Συστήματα Διαχείρισης Βάσεων Δεδομένων και με την βοήθεια τους μπορούμε να εκτελέσουμε όλες τις παραπάνω λειτουργίες με ευκολία.

Οι βασικότερες κατηγορίες βάσεων δεδομένων περικλείονται στις εξής:

- Σχεσιακές
- Αντικειμενο-σχεσιακές
- Αντικειμενοστρεφείς

Κυρίαρχος τύπος βάσεων δεδομένων που χρησιμοποιούνται σήμερα είναι οι σχεσιακές οι οποίες χάρην στην γλώσσα προγραμματισμού SQL (Structured Query Language) μας δίνουν την δυνατότητα να χρησιμοποιήσουμε τα ερωτήμα που έχουμε συγγράψει άσχετα από το ποια γλώσσα προγραμματισμού θα επιλέξουμε για την εφαρμογή μας.

Σε αυτήν την πτυχιακή όμως θα ασχοληθούμε με αντικειμενοστρεφείς βάσεις δεδομένων και συγκεκριμένα με την ObjectDB. Γιατί όμως τις επιλέγουμε ενώ οι σχεσιακές είναι πρώτη προτίμηση της πλειοψηφίας των εφαρμογών; Απλούστατα γιατί οι σχεσιακές βάσεις δεδομένων δεν μπορούν να αντιληφθούν τα αντικείμενα και τις κλάσεις όπως οι αντικειμενοστρεφείς. Οπότε με αυτόν τον τρόπο επιτυγχάνουμε αρμονία και σε αυτά τα δύο μέρη της πτυχιακής.

Η γλώσσα προγραμματισμού Java και το NetBeans

Στις αρχές της δεκαετίας του 90 η Sun Microsystems δημιούργησε την γλώσσα προγραμματισμού Java προκειμένου να την βοηθήσει στην ανάπτυξη λογισμικού για μικροσυσκευές. Ένα από τα βασικά χαρακτηριστικά της είναι η ανεξαρτησία του λειτουργικού συστήματος που τη φιλοξενεί και το οποίο έλυσε η εικονική μηχανή της Java. Όταν πρόκειται να εκτελέσουμε μία εφαρμογή γραμμένη σε Java το Java Virtual Machine πρέπει να είναι εγκατεστημένο στο λειτουργικό σύστημα. Αυτό θα αναλάβει να διαβάσει τα αρχεία .class που δημιουργεί ο μεταγλωττιστής javac και στην συνέχεια θα τα μεταφράσει στην γλώσσα μηχανής που αντιλαμβάνεται ο εκάστοτε επεξεργαστής. Επομένως αρκεί να αναπτύξουμε μόνο μία έκδοση της εφαρμογής μας η οποία θα μπορει να εκτελεστεί σε κάθε επεξεργαστή και κάθε λειτουργικό σύστημα.

Η ραγδαία εξάπλωση του Internet και του World-Wide Web δημιούργησαν την ανάγκη νέων τρόπων ανάπτυξης και διανομής του λογισμικού. Η Java σχεδιάστηκε με σκοπό την ανάπτυξη εφαρμογών που θα τρέχουν σε ετερογενή δικτυακά περιβάλλοντα.

Η Java έχει τα ακόλουθα χαρακτηριστικά

• Αντικειμενοστρεφής (ομοιότητες εντολών με τη C++).

• Δημιουργία ανεξάρτητων εφαρμογών και applets (applet = προγράμματα που περιλαμβάνονται σε HTML σελίδες και εκτελούνται από τον Web Browser).

• Είναι Interpreted γλώσσα. Αυτό σημαίνει ότι ο java compiler δεν παράγει εκτελέσιμο κώδικα αλλά μια μορφή ψευδοκώδικα (bytecode) το οποίο από μόνο του δεν τρέχει σε καμία μηχανή. Προκειμένου λοιπόν να εκτελεστεί απαιτείται η χρήση ενός interpreter(=διερμηνέα) για να μετατρέψει το bytecode σε πραγματικό εκτελέσιμο κώδικα. Αυτό το χαρακτηριστικό δίνει τη δυνατότητα στα java bytecodes να μπορούν να τρέξουν σε οποιοδήποτε μηχάνημα, κάτω από οποιοδήποτε λειτουργικό, αρκεί να έχει εγκατασταθεί ένας java interpreter. Επίσης, ένα άλλο χαρακτηριστικό του java bytecode είναι το μικρό του μέγεθος, (μόλις λίγα Kilobytes). Αυτό το κάνει ιδανικό για μετάδοση μέσω του δικτύου.

• Κατανεμημένη (distributed). Δηλαδή, ένα πρόγραμμα σε Java είναι δυνατό να το φέρουμε από το δίκτυο και να το τρέξουμε. Επίσης είναι δυνατό διαφορετικά κομμάτια του προγράμματος να έρθουν από διαφορετικά sites.

• Ασφαλής (secure). Στο δίκτυο όμως ελλοχεύουν πολλοί κίνδυνοι για τον χρήστη - παραλήπτη μιας δικτυακής εφαρμογής, γι' αυτό η Java έχει σχεδιαστεί έτσι ώστε να ελαχιστοποιείται η πιθανότητα προσβολής του συστήματος του χρήστη από κάποιο applet γραμμένο για τέτοιο σκοπό.

• Είναι multithreaded. Η Java υποστηρίζει εγγενώς την χρήση πολλών threads. Προκειμένου να το επιτύχει αυτό σε συστήματα με έναν επεξεργαστή, το Java runtime system (interpreter) υλοποιεί ένα δικό του χρονοδρομολογητή (scheduler), ενώ σε συστήματα που υποστηρίζουν πολυεπεξεργασία η δημιουργία των threads ανατίθεται στο λειτουργικό σύστημα. Φυσικά όλα αυτά είναι αόρατα τόσο στον προγραμματιστή όσο και στον χρήστη.

• Υποστηρίζει multimedia εφαρμογές. Με αυτό εννοούμε ότι η Java παρέχει ευκολίες στη δημιουργία multimedia εφαρμογών. Αυτό επιτυγχάνεται τόσο με την ευελιξίας της σαν γλώσσα όσο και με τις πλούσιες και συνεχώς εμπλουτιζόμενες βιβλιοθήκες της.

Ο βασικότερος λόγος που επιλέχθηκε αυτή η γλώσσα προγραμματισμού είναι γιατί η εφαρμογή θα πρέπει να μπορεί να εκτελεστεί σε διαφορετικά

λειτουργικά συστήματα και επίσης γιατί αυτή είναι η γλώσσα που χρησιμοποιείται περισσότερο στην σχολή μας.

Για την ανάπτυξη της διεπαφής της εφαρμογής χρησιμοποιήσαμε την βιβλιοθήκη Java Swing η οποία είναι η κύρια βιβλιοθήκη ανάπτυξης GUI εφαρμογών (δηλαδή εφαρμογών με γραφικό περιβάλλον).

Για να γράψει κάποιος κώδικα σε Java αρκεί μόνο ένας απλός επεξεργαστής κειμένου. Ωστόσο, υπάρχουν διάφορα ολοκληρωμένα περιβάλλοντα ανάπτυξης λογισμικου (IDE) σε Java που βοηθούν πολύ στην ταχύτητα ανάπτυξης τους λογισμικού και την οργάνωση του κώδικα. Εμείς χρησιμοποιήσαμε το IDE NetBeans.

Το NetBeans ξεκίνησε το 1996 ως ένα ερευνητικό έργο ενός φοιτητή στο Charles University της Πράγας και είναι γραμμένο σε Java. Μετά από 3 χρόνια αγοράστηκε από την Sun Microsystems (που ανήκει πλέον στην Oracle) και σήμερα είναι το κυρίαρχο Java IDE μαζί με το Eclipse. Με τις αυτοματοποιημένες λειτουργίες που διαθέτει προσφέρει μία αρκετά απλοποιημένη λύση για την ανάπτυξη λογισμικού, ειδικά σε Java Swing. Το NetBeans επιλέχθηκε κυρίως λόγω εμπειρίας στην χρησιμοποίηση του.

Η αντικειμενοστρεφής βάση δεδομένων ObjectDB

Η Objectdb είναι μία από τις αντικειμεντοστρεφείς βάσεις δεδομένων που είναι υλοποιημένες σε Java. Είναι ιδιαίτερα ευέλικτη καθως μπορεί να χρησιμοποιηθεί σε client-server mode και ως ενσωματτωμένη στην εφαρμογή. Χρησιμοποιεί ένα από τα JPA και JDO τα οποία είναι έτοιμα Java APIs οπότε δεν απαιτεί ενδιάμεσο ORM (Object-Relational Mapping) λογισμικό, αλλά δεν έχει υποστήριξη SQL. Στην εφαρμογή μας χρησιμοποιούμε το JPA API.

Καθώς γραμμένη σε Java, η ObjectDB μπορεί να λειτουργήσει σε οποιοδήποτε λειτουργικό σύστημα με Java SE 5 ή μεγαλύτερο. Το μέγιστο μέγεθος της βάσης δεδομένων είναι 128 TB με απεριόριστο αριθμό αντικειμένων. Παρόλαυτα η δωρεάν έκδοση της περιορίζει τον αριθμό κλάσεων (Entity Classes) που αποθηκεύονται σε 10. Για να δημιουργήσουμε περισσότερα διαφορετικού

είδους αντικείμενα χρειάζεται να αγοραστεί η άδεια του λογισμικού. Στην δική μας περίπτωση υλοποιούνται 9 Entity Classes, οι

- Athlete
- Judge
- Trainer
- Volunteer
- Sport
- Game
- Stadium
- Participation
- Help

Κάθε αντικείμενο στην ObjectDB έχει μοναδικό αναγνωριστικό (ID). Υποστηρίζει εκτός από τις παραδοσιακά ID των Αντικειμενοστρεφών Βάσεων Δεδομένων και τα κλειδιά των Σχεσιακών βάσεων καθώς επίσης και τα σύνθετα κλειδιά.

Το πρώτο βήμα για να ενσωματτώσουμε την ObjectDB στο NetBeans Project είναι να εισάγουμε την βιβλιοθήκη του σε αυτό (μπορούμε να την κατεβάσουμε από εδώ)

Για να ορίσουμε μία κλάση ως έγκυρη JPA Entity Class πρέπει

να δηλώσουμε με Annotation πάνω από τον ορισμό της κλάσης το

@Entity

να κάνουμε import τα java.io.Serializable και javax.persistence.*

import java.io.Serializable;

import javax.persistence.;*

@Entity

public class Athlete implements Serializable

{

Επίσης πρέπει η κλάση μας να υλοποιεί το Interface Serializable

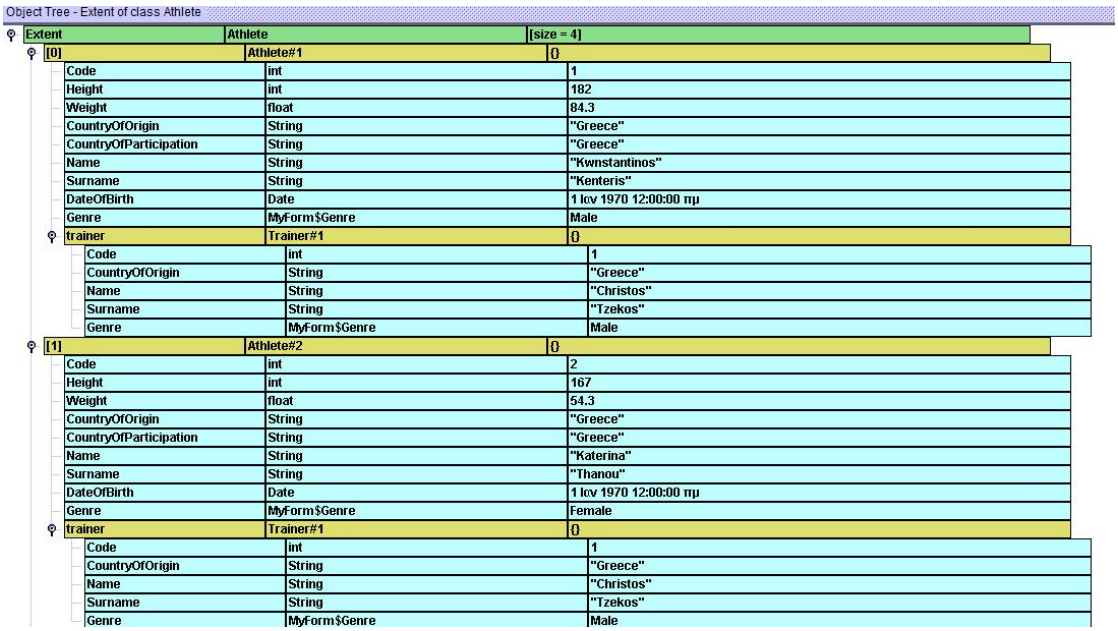
Στην συνέχεια ορίζουμε την Main Class στην οποία χρειαζόμαστε τα packages javax.persistence και java.util.*

Με τον παρακάτω κώδικα δημιουργούμε σύνδεση στην βάση δεδομένων μας

```
    EntityManagerFactory emf =
                                    Persistence.createEntityManagerFactory
("LinkedData.odb");
    EntityManager em = emf.createEntityManager();
```

Έπειτα μπορούμε να εισάγουμε αντικείμενα στην βάση

```
    em.getTransaction().begin();
    Stadium st1 = new Stadium(1, "Marakana", 120000, "Rio De Janeiro");
    em.persist(st1);
    em.getTransaction().commit();
```



Σχήμα 1

Όλα τα δεδομένα αποθηκεύονται σύμφωνα με το συγκεκριμένο παράδειγμα σε ένα αρχείο (LinkedData.odb) στο root directory του Project. Από εκεί μπορούμε να ανακτήσουμε τα δεδομένα είτε όλα μαζί ως Objects είτε κάθε τύπο ξεχωριστά.

```
TypedQuery<Object> query = em.createQuery("SELECT o FROM Object o",
Object.class);

        List<Object> results = query.getResultList();
```

Τέλος χρειάζεται απλά να κλείσουμε την σύνδεση που δημιουργήσαμε νωρίτερα με τις εντολές em.close(); και emf.close();

Σημασιολογικός Ιστός (Semantic Web)

Καθώς το διαδίκτυο μεγαλώνει και επεκτείνεται γεννιούνται καθημερινά νέες τεχνολογίες και εφευρίσκονται νέοι τρόποι παρουσίασης και επεξεργασίας δεδομένων. Ο Σημασιολογικός Ιστός (Semantic Web) είναι μία επέκταση του ήδη υπάρχοντα ιστού ο οποίος βασίζει την λειτουργία του στην σημασία των δεδομένων. Προσπαθεί να επεξεργαστεί και να συνδέσει τα δεδομένα που έχει

διαθέσιμα χρησιμοποιώντας μετα-δεδομένα (metadata) ώστε και οι χρήστες αλλά και οι μηχανές να τα «κατανοούν». Τι θα καταφέρουμε όμως με αυτόν τον τρόπο;

Ο βασικός στόχος του Σημασιολογικού Ιστού είναι να δώσει την δυνατότητα στους χρήστες να βρούν, διαμοιράσουν και να συνδυάσουν δεδομένα όσο το δυνατόν πιο εύκολα. Με αυτόν τον τρόπο θα βοηθήσουμε τις μηχανές να εκτελέσουν ενέργειες τις οποίες θα ήταν αδύνατο να εκτελέσουν μόνες τους, χωρίς την καθοδήγηση ενός χρήστη. Οι μηχανές είναι πολύ αποδοτικές στο να βρίσκουν δεδομένα. Όταν όμως η δομή τους είναι διακριτή δεν μπορούν να τα εκμεταλλευτούν όπως ο άνθρωπος. Συνδέοντας αυτά τα δεδομένα μεταξύ τους μπορούμε να τις δώσουμε τις κατευθύνσεις για να βρίσκουν «νόημα» σε αυτά και να καταλαβαίνουν τις απαιτήσεις μας.

Ο Ιστός (Web) του μέλλοντος προβλέπεται να αποτελεί μια παγκόσμια βάση δεδομένων και γνώσης με πληροφορίες οι οποίες θα είναι "κατανοητές" από μηχανές (machine-understandable information). Οι κύριες τεχνολογίες για την υλοποίηση του Σημασιολογικού Ιστού είναι ο σημασιολογικός εμπλουτισμός και η χρήση των οντολογιών.

Η λέξη "Σημασιολογία" έχει ρίζα τις Ελληνικές λέξεις "σημάδι", "σημαίνω" και "σημαντικός" και σήμερα αναφέρεται στο νόημα συχνά σε επίπεδο γλώσσας. Μπορούμε να πούμε ότι ο Σημασιολογικός Ιστός αποτελεί το μεγαλύτερο σε παγκόσμιο επίπεδο έργο ευφυής ενσωμάτωσης συστημάτων ώστε να συνεργάζονται δια-λειτουργικά.

Ο Tim Berners-Lee, που επινόησε τον Παγκόσμιο Ιστό το 1989, είχε το όραμα, που τώρα συμμερίζονται πολλοί - ενός ιστού δεδομένων που μπορούν να επεξεργαστούν από μηχανές.

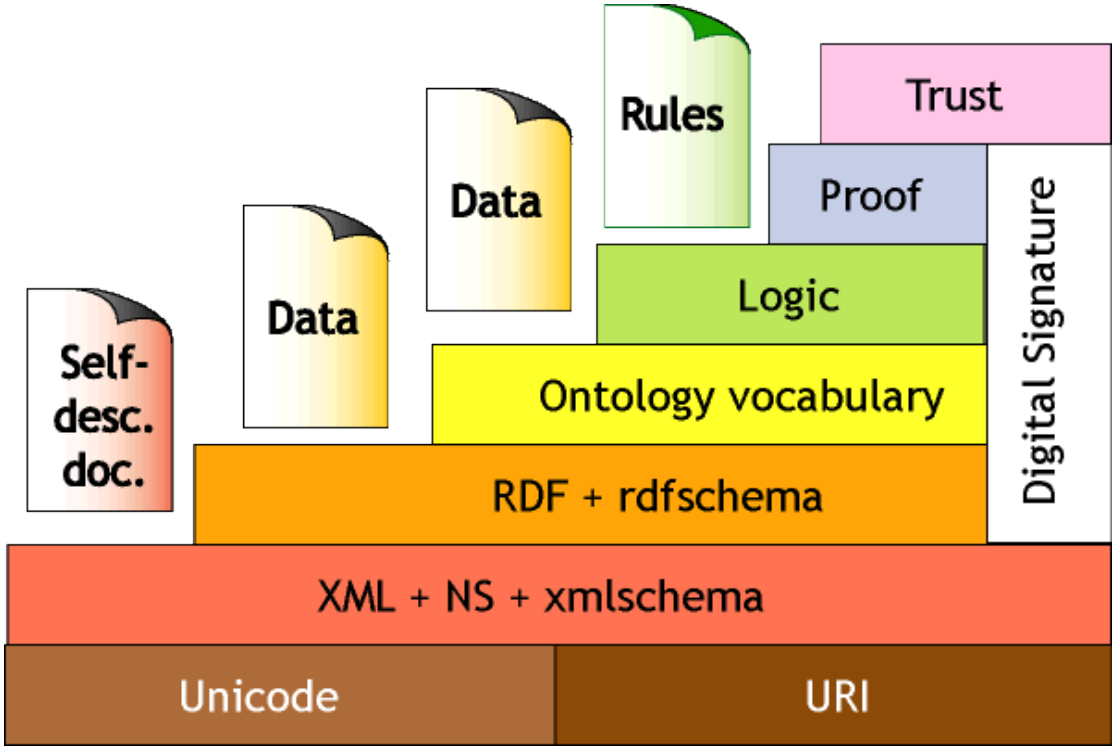
"Ο Σημασιολογικός Ιστός είναι μια επέκταση του σημερινού ιστού όπου η πληροφορία έχει καλά καθορισμένο νόημα, καθιστώντας τη συνεργασία μεταξύ ανθρώπων και υπολογιστών πιο αποτελεσματική" [1].

Το κέντρο βάρους του περιεχομένου του Ιστού μετατοπίζεται συνεχώς από τον άνθρωπο προς προς τα δεδομένα. Για να φτάσει ο Ιστός το μέγιστο των δυνατοτήτων του, πρέπει να εξελιχθεί σε ένα Σημασιολογικό Ιστό, ο οποίος

παρέχει μια διεθνώς προσβάσιμη πλατφόρμα που επιτρέπει σε αυτοματοποιημένα εργαλεία αλλά και σε ανθρώπους να μοιράζονται και να επεξεργάζονται δεδομένα.

Η κυρίαρχη λογική πίσω από τον Σημασιολογικό Ιστό είναι ότι κάθε πληροφορία περιέχει μετα-δεδομένα. Αυτά υλοποιήθηκαν ώστε να δίνουν πληροφορίες στους υπολογιστές για κάθε δεδομένο που δεν μπορούν να έχουν αλλιώς. Όσο καιρό ο όγκος των αποθηκευμένων πληροφοριών παρέμενε σε χαμηλά επίπεδα, ο key-word τρόπος άντλησης τους ήταν αποτελεσματικός. Με τη ραγδαία όμως αύξησή του οι παρενέργειες δεν άργησαν να φανούν. Παρόλο που η αναζήτηση μιας λέξης θα μας αποφέρει χιλιάδες άμεσα συσχετιζόμενες πληροφορίες, η ύπαρξη επίσης χιλιάδων πληροφοριών που δεν σχετίζονται άμεσα με το πεδίο της αναζήτησης μας αλλά τυγχάνει να περιέχουν τη λέξη κλειδί που έχουμε χρησιμοποιήσει, δημιουργούν ένα φόρτο γνώσης, που ο μέσος χρήστης είναι συχνά ανήμπορος να διαχειριστεί αποτελεσματικά.

Το semantic web δεν θα αποτελεί έναν παγκόσμιο ιστό που θα λειτουργεί παράλληλα με τον ήδη υπάρχων ιστό, αλλά θα αποτελεί μια μετεξέλιξη του. Πρώτο στάδιο για την δημιουργία του έργου αυτού, η αναπαράσταση της γνώσης σε machine accessible γλώσσα. Ένας τρόπος για να πραγματοποιηθεί αυτό είναι τα Linked Data, ζήτημα που θα αναλυθεί στην ενότητα που ακολουθεί.



Σχήμα 2

Ο Σημασιολογικός Ιστός αποτελεί πρωτοβουλία της Κοινοπραξίας του Παγκοσμίου Ιστού (W3C) και η σχετική Δραστηριότητα (W3C Semantic Web Activity) έχει δημιουργηθεί για να εξυπηρετήσει έναν ηγετικό ρόλο, τόσο στο

σχεδιασμό προδιαγραφών, όσο και στην ανοικτή ανάπτυξη της τεχνολογίας μέσω της συνεργασίας.

Διασυνδεδεμένα Δεδομένα (Linked Data)

Τα διασυνδεδεμένα δεδομένα είναι μία μέθοδος δημοσιοποίησης δομημένων δεδομένων που βασικός στόχος της είναι να ενώσει φαινομενικά «άσχετα» δεδομένα μεταξύ τους βρίσκοντας τι πραγματικά τα συνδέει. Χρησιμοποιεί τεχνολογίες ιστού όπως το HTTP και URIs ώστε να ανταλλάσει πληροφορίες μεταξύ των ιστοσελίδων με τρόπο επεξεργάσιμο από τους υπολογιστές. Σε αυτό το σημείο πρέπει να κατανοήσουμε ότι ο Σημασιολογικός Ιστός και κατ' επέκταση τα Διασυνδεδεμένα Δεδομένα δεν αναπτύχθηκαν για να αλλάξουν το πώς βλέπουμε και κατανοούμε εμείς οι άνθρωποι τις ιστοσελίδες, αλλά για να δώσουν αυτό το δικαίωμα και στις μηχανές με απώτερο σκοπό να μας βοηθήσουν να συνδέσουμε τις πληροφορίες που υπάρχουν διαθέσιμες και να διευκολύνουν την επεξεργασία τους.

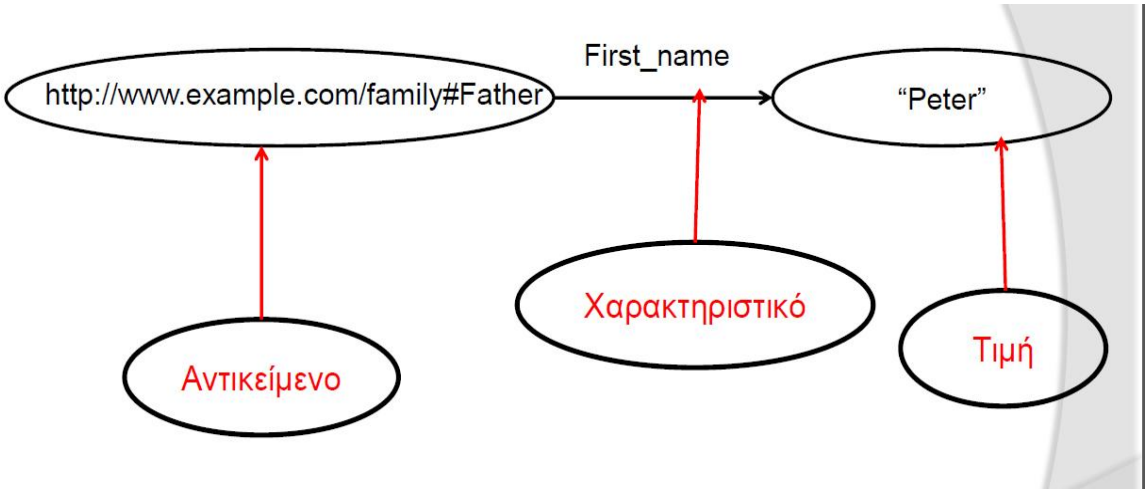
Τα Linked Data αναπτύχθηκαν και προωθήθηκαν από τον Tim Berners Lee (διευθυντή του World Wide Web Consortium – W3C) ο οποίος όρισε τέσσερις κανόνες:

1. Χρησιμοποίησε URIs ως ονόματα για δεδομένα
2. Χρησιμοποίησε HTTP URIs έτσι ώστε οι άνθρωποι να μπορούν να διαβάζουν αυτά τα ονόματα
3. Δώσε χρήσιμες πληροφορίες σχετικά με το θέμα όταν διαβάζεται αυτό το URI χρησιμοποιώντας κάποιες standard τυποποιήσεις (όπως RDF, SPARQL)
4. Συμπεριέλαβε συνδέσεις σε άλλα URIs ώστε να μπορούν να ανακαλύψουν κι άλλες σχετιζόμενες πληροφορίες ακόμα κι αν δεν γνωρίζουν για αυτές.

Σχήμα 3

RDF

Το RDF είναι μία γλώσσα περιγραφής πόρων (metadata) και αναπαράστασης γνώσης για τον Σημασιολογικό Ιστό. Με την έννοια πόρος αναφερόμαστε σε οποιαδήποτε οντότητα του Παγκόσμιου Ιστού, όπως είναι μία ιστοσελίδα, ένα τμήμα της ή ένα σύνολο από ιστοσελίδες, ηλεκτρονικά αρχεία κτλ. Βασίζεται στην ιδέα του να δημιουργούνται δηλώσεις σχετικά με δεδομένα με την μορφή Υποκείμενο – Ιδιότητα – Αντικείμενο. Αυτού του είδους οι εκφράσεις είναι γνωστές ως Τριάδες (Triples). Το υποκείμενο δηλώνει το δεδομένο και η ιδιότητα περιγράφει την σχέση ή σύνδεση μεταξύ του υποκειμένου και του αντικειμένου. Για παράδειγμα η φράση «Ο ουρανός είναι μπλε» μπορεί να αναπαρασταθεί ως Triple με τον «ουρανό» στην θέση του υποκειμένου, το «είναι» ως ιδιότητα και το «μπλε» ως το αντικείμενο.



Σχήμα 4

Ένα σύνολο από τριάδες RDF μπορούμε να το αντιληφθούμε και ως έναν γράφο. Σε αυτόν τον γράφο τα αντικείμενα και τα υποκείμενα παίζουν το ρόλο των κόμβων ενώ οι ιδιότητες παίζουν το ρόλο των ακμών που τους συνδέουν. Όπως αναφέραμε και στην εισαγωγή, η RDF είναι μια γλώσσα αναπαράστασης γνώσης για το Σημασιολογικό Ιστό. Έτσι λοιπόν το πρότυπο της RDF καθορίζει και μια σύνταξη η οποία έχει σαν σκοπό οι RDF τριάδες να δομούνται με έναν τρόπο επεξεργάσιμο από υπολογιστικά συστήματα και εφαρμογές. Η σύνταξη αυτή δε θα μπορούσε να βασίζεται σε άλλο πρότυπο παρά μόνο στην XML. Η σύνταξη αυτή αναφέρεται ως RDF/XML σύνταξη. Για λόγους γενικότητας η RDF χρησιμοποιεί URI references για να προσδιορίσει τις οντότητες οι οποίες βρίσκονται στη θέση του υποκειμένου, της ιδιότητας και του αντικειμένου. Ένα URI reference (URIRef) αποτελείται από ένα URI και από ένα προαιρετικό fragment identifier. Για παράδειγμα το URIRef http://www.example.org/index.html#section2 αποτελείται από το URI http://www.example.org/index.html και από τον fragment identifier Section2 τον οποίο διακρίνουμε από το URI με τη χρήση του συμβόλου #. Όπως γίνεται αντιληπτό ίσως είναι λίγο άβολο να γράφουμε συνέχεια ολόκληρα τα URI των οντοτήτων που χρησιμοποιούμε. Έτσι λοιπόν πολλές φορές χρησιμοποιούνται συντομεύσεις σε μορφή προθεμάτων (prefixes) τα οποία ονομάζονται QNames.

Υπάρχουν διάφορα Serialization Formats όπως

- Turtle
- N-Triples
- N-Quads
- JSON-LD
- N3

- RDF/XML

Εμείς θα χρησιμοποιήσουμε την μορφή Turtle που είναι μία απλή συμπαγής και φιλική προς τον χρήστη μορφοποίηση

Turtle

Η Turtle είναι μία τυποποιημένη μορφή έκφρασης δεδομένων στο RDF μοντέλο σύνταξης. Παρέχει την δυνατότητα ομαδοποίησης τριπλετών και τρόπους για να συντομεύουμε την δήλωση παρόμοιων αντικειμένων ή ιδιοτήτων.

Στην αρχή του εγγράφου δηλώνουμε τις συντομεύσεις ως

@prefix Athlete: http://it.teithe.gr/Rdf/Turtle/Athlete/

Τα οποία χρησιμοποιούμε στις δηλώσεις των Triples ως εξής

<Athlete:1>

:Name "Kwnstantinos" ;

:Surnname "Kenteris" ;

:Genre "Male" ;

:DateOfBirth "1970-01-01" ;

:Weight 84.3 ;

:Height 182 ;

:CountryOfOrigin "Greece" ;

:CountryOfParticipation "Greece" .

Στις αγκύλες δηλώνουμε το URI στο οποίο αναφερόμαστε με την συντόμευση του (σε αυτήν την περίπτωση “*Athlete:*”) και στην συνέχεια προσθέτουμε το υποκείμενο, που είναι το ID του αθλητή, δηλαδή το “1”. Μετά ακολουθούν οι

ιδιότητες αυτού του υποκειμένου. Ορίζουμε ως Name το “Kwnstantinos” κτλ. Κάθε δήλωση Triple τελειώνει με την τελεία “.”.

ΚΕΦΑΛΑΙΟ 1 – Δημιουργία Αντικειμενοστρεφούς Βάσης Δεδομένων OlympicGames

ΕΙΣΑΓΩΓΗ

Σε αυτό το κεφάλαιο θα εξηγήσουμε πως δημιουργήθηκε η Αντικειμενοστρεφής Βάση Δεδομένων μας. Ως πρότυπο χρησιμοποιήθηκε η Βάση Δεδομένων «Ολυμπιακοι Αγώνες» που μελετήσαμε στο μάθημα Βάσεις Δεδομένων ΙΙ (το σχεσιακό σχήμα της και τα SQL Statements δημιουργίας της συμπεριλαμβάνονται στο CD της πτυχιακής).

Αυτή η βάση περιέχει εννέα (9) πίνακες οι οποίοι παρουσιάζονται παρακάτω επιγραμματικά:

- Εθελοντής (Αστυνομική Ταυτότητα, Όνομα, Επώνυμο, Φύλο, Ηλικία)
- Στάδιο (Όνομα, Χωρητικότητα, Πόλη)
- Κριτής (Κωδικός, Όνομα, Επώνυμο, Φύλο, Ημερομηνία Γέννησης)
- Προπονητής (Κωδικός, Όνομα, Επώνυμο, Χώρα Καταγωγής)
- Άθλημα (Κωδικός, Όνομα, Φύλο, Ρεκόρ, Ημερομηνία Ρεκόρ, Επώνυμο Ολυμπιονίκη, Όνομα Ολυμπιονίκη)
- Αθλητής (Κωδικός, Όνομα, Επώνυμο, Φύλο, Ημερομηνία Γέννησης, Βάρος, Ύψος, Χώρα Καταγωγής, Χώρα με την οποία συμμετέχει, Κωδικός Προπονητή)
- Αγώνες (Κωδικός, Επίπεδο, Ημερομηνία Αγώνα, Ώρα Αγώνα, Όνομα Σταδίου, Κωδικός Αθλήματος, Κωδικός Κριτή)
- Βοηθάει (Κωδικός Αγώνα, Ταυτότητα Εθελοντή)
- Συμμετέχει (Κωδικός Αγώνα, Κωδικός Αθλητή, Καλύτερη Επίδοση, Έγκυρος ή Ακύρος)

1.1 Δημιουργία Project στο NetBeans

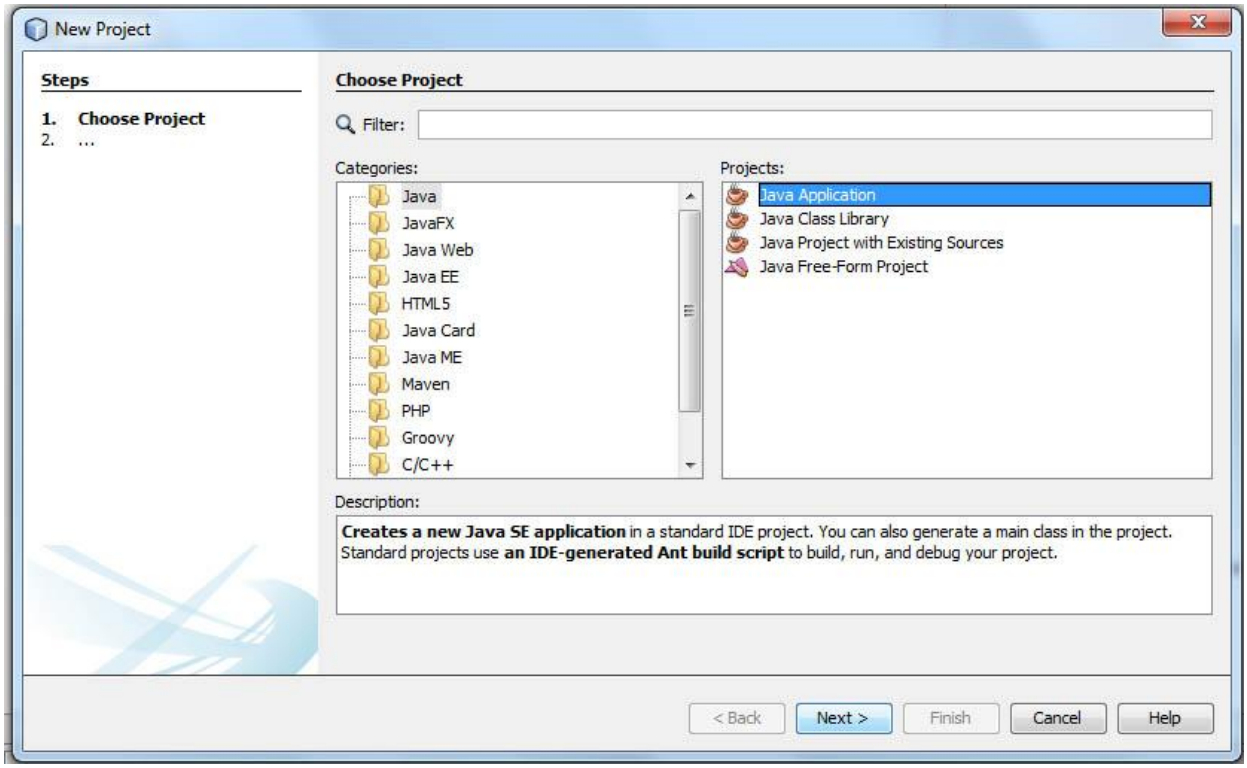
Το πρώτο βήμα του πρακτικού μέρους της πτυχιακής είναι η δημιουργία του Project στο οποίο αναπτύχθηκε η εφαρμογή. Ανοίγουμε το NetBeans και επιλέγουμε

File ➔ New Project

Από τις επιλογές που μας παρουσιάζει το IDE επιλέγουμε το

Java ➔ Java Application

Όπως φαίνεται και στην εικόνα



Σχήμα 5

Έπειτα δίνουμε όνομα στην εφαρμογή μας και φροντίζουμε να μην είναι επιλεγμένο το Create Main Class. Αυτό το αποφεύγουμε γιατί θα αναπτύξουμε μία εφαρμογή με διεπαφή χρήστη. Τέλος πατάμε Finish.

Πλέον έχουμε φτιάξει το Project μας και είμαστε έτοιμοι να δημιουργήσουμε τις κλάσεις σύμφωνα με το σχήμα της βάσης των Ολυμπιακών Αγώνων.

Επόμενο βήμα είναι να εισάγουμε την βιβλιοθήκη της ObjectDB όπως φαίνεται παρακάτω

Σχήμα 6

Σημείωση: Δεν χρειάζεται να εισάγουμε και τις βιβλιοθήκες των JPA ή JDO καθώς η objectdb τις περιλαμβάνει.

1.2 Δημιουργία Serializable κλάσεων

Το JPA API ορίζει μία σειρά από Annotations για την δημιουργία Entity κλάσεων, δηλαδή κλάσεων αποδεκτών για να αποθηκευτούν στην ObjectDB. Η αρχική μορφή της κλάσης Athlete είναι η παρακάτω:

```
import java.sql.Date;

public class Athlete
{
   int Code;
   String Name;
   String Surname;
   MyForm.Genre Genre;
   Date DateOfBirth;
   float Weight;
   int Height;
   String CountryOfOrigin;
   String CountryOfParticipation;

   Trainer trainer;

   // Default constructor
   public Athlete() {}
}
```

Κάνουμε import δύο packages, τα

- java.io.Serializable και
- javax.persistence.

Το πρώτο μας επιτρέπει να κάνουμε Serialize την κλάση μας ώστε να χειριστούμε τα δεδομένα της με όποιον τρόπο επιλέξουμε. Το δεύτερο περιέχει τις δηλώσεις που θα χρησιμοποιήσουμε για να την δηλώσουμε ως Entity Class και να δηλώσουμε και τα Attributes της, όπως το ID και τις σχέσεις που έχει με άλλες κλάσεις, στην συγκεκριμένη περίπτωση την κλάση Trainer.

```
@Entity
public class Athlete implements Serializable
```

Το Annotation @Entity δηλώνει ότι η συγκεκριμένη κλάση είναι μία user defined κλάση που θα αποθηκευτεί στην βάση δεδομένων.

```
    @Id
    int Code;
```

Το Annotation @Id δηλώνει ότι το πεδίο Code είναι το αναγνωριστικό του αντικειμένο. Αυτό πρέπει να είναι μοναδικό για κάθε Athlete. Αν προσθέσουμε το @GeneratedValue τότε η αρίθμηση του δίνεται από την ObjectDB, ειδάλλως είμαστε υποχρεωμένοι να του δώσουμε μοναδική τιμή από την εφαρμογή. Ως ID μπορούν να δηλωθούν :

- όλοι οι βασικοί τύποι της Java
- οι αντίστοιχες wrapper classes του κάθε τύπου από το package java.lang (π.χ. Integer, Double κτλ)
- java.Math.BigInteger και java.Math.BigDecimal
- java.lang.String
- java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp
- Οποιοσδήποτε τύπος Enum
- Αναφορές σε Entity αντικείμενα

Επίσης μπορούμε να δηλώσουμε σύνθετα κλειδιά με το Annotation @Id μπροστά από κάθε πεδίο.
Στην περίπτωση του Αθλητή, υπάρχει ως πεδίο και ένας προπονητής (Trainer). Σε μία σχεσιακή βάση θα το δηλώναμε ως εξωτερικό κλειδί το οποίο θα συνδεόταν με

το ID του προπονητή. Το JPA API μας επιτρέπει να δηλώσουμε με παρόμοιο τρόπο αυτήν την σχέση τοποθετώντας το Annotation

@JoinColumn(name="Trainer_Code")

Πάνω από την εντολή

Trainer trainer;

Έτσι κάνουμε “Join” τα αντικείμενα τύπου Athlete με τα Trainer περνώντας μία αναφορά του αντικειμένου Trainer στο πεδίο Athlete.trainer. Στο γραφικό περιβάλλον που μας παρέχει η ObjectDB μπορούμε να προσπελάσουμε όλα τα πεδία του trainer μέσω του Athlete. Βεβαίως, όπως και στις σχεσιακές βάσεις, πρέπει να δηλώσουμε τον τύπο της σχέσης που συνδέει Αθλητή και Προπονητή. Οι τύποι είναι οι ακόλουθοι:

- @OneToOne
- @ManyToOne
- @OneToMany

Σε αυτήν την περίπτωση η σχέση είναι ManyToOne. Δηλαδή πολλοί αθλητές μπορούν να έχουν τον ίδιο προπονητή, αλλά κάθε αθλητής έχει έναν προπονητή.

1.3 Persistence.xml

Το Persistence.xml είναι απαραίτητο για την υλοποίηση κώδικα με την βοήθεια του JPA API. Το τι θα περιέχει όμως είναι στο χέρι μας. Βρίσκεται στον υποφάκελο META-INF και μπορούμε να συμπληρώσουμε μέσα σε αυτό persistent units με τις εξής υποκατηγορίες:

- Τον Provider που υποδηλώνει ποια JPA υλοποίηση πρέπει να χρησιμοποιηθεί
- Το mapping-file element μας επιτρέπει να ορίσουμε XML αρχεία με annotations
- Το jar-file element ορίζουν αρχεία jar τα οποία πρέπει να ελεγχούν για managed persistent classes
- Το class element για να δώσουμε μία λίστα με τα ονόματα των κλάσεων
- Το property element για διάφορες ιδιότητες όπως το URL της βάσης δεδομένων, username και password κτλ

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="OlympicGamesPU" transaction-
type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>Athlete</class>
    <class>Game</class>
    <class>Helps</class>
    <class>Judge</class>
    <class>Participates</class>
    <class>Sport</class>
    <class>Stadium</class>
    <class>Trainer</class>
    <class>Volunteer</class>
    <properties>
      <property name="javax.persistence.jdbc.url"
value="jdbc:derby://localhost:1527/sample"/>
      <property name="javax.persistence.jdbc.password" value="app"/>
      <property name="javax.persistence.jdbc.driver"
value="org.apache.derby.jdbc.ClientDriver"/>
      <property name="javax.persistence.jdbc.user" value="app"/>
      <property name="eclipselink.ddl-generation" value="drop-and-create-
tables"/>
      <property name="eclipselink.logging.exceptions" value="true"/>
    </properties>
  </persistence-unit>
</persistence>
```

1.4 Εισαγωγή Αντικειμένων στην Βάση Δεδομένων

Έχουμε εξηγήσει πως δημιουργούνται οι Entity κλάσεις που θέλουμε να εισάγουμε στην Βάση Δεδομένων. Η δημιουργία στιγμιοτύπων αυτών των κλάσεων είναι απλή διαδικασία όπως γνωρίζουμε από τα πρώτα μαθήματα προγραμματισμού στην σχολή μας. Πως όμως θα τα αποθηκεύσουμε αυτά τα αντικείμενα στην ObjectDB; Το JPA API μας παρέχει έναν πολύ απλό τρόπο για να το επιτύχουμε. Στην αρχή της κλάσης της εφαρμογής που περιέχει την μέθοδο main() έχουμε δηλώσει το EntityManager em; Αυτό το αντικείμενο του interface EntityManager περιέχει διάφορες μεθόδους όπως οι getTransaction() και persist(Object obj) που μας ενδιαφέρουν. Πρώτα πρέπει να ορίσουμε ότι η κατάσταση της σύνδεσης μας στην βάση δεδομένων είναι ανοικτή σε εισαγωγή ή επεξεργασία Entity untis με την παρακάτω εντολή

```
em.getTransaction().begin();
```

Αφού δημιουργήσουμε ένα αντικείμενο με τον παραδοσιακό τρόπο της Java η αρχική του κατάσταση είναι New. Μία κλήση στο persist συνδέει το αντικείμενο με ένα EntityManager, στην προκειμένη περίπτωση τον em, και αλλάζει την κατάσταση του από New σε Managed. Το νέο Entity αντικείμενο αποθηκεύεται στην βάση όταν κάνουμε commit το Transaction.

```
Athlete a1 = new Athlete();
em.persist(a1);
em.getTransaction().commit();
```

Ποια είναι όμως η λειτουργία του Transaction; Είναι παρόμοια με αυτήν που έχει στα RDBM συστήματα. Αν υπάρξει κάποιο σφάλμα πριν κατατεθεί (commit) το σύνολο των εντολών που δώσαμε στην συναλλαγή μας (Transaction), δεν θα προχωρήσει καμία καταχώρηση στην βάση και θα γίνει rollback. Δηλαδή η εντολή getTransaction().begin() μας προστατεύει από την ελλιπή και λανθασμένη καταχώρηση δεδομένων.

Όπως καταλαβαίνουμε είναι συνετό να δημιουργούμε ξεχωριστά block εισαγωγών με σχετικές μεταξύ τους καταχωρήσεις αντικειμένων ώστε να προστατέψουμε την ακεραιότητα των δεδομένων μας. Για τις ανάγκες της εφαρμογής δημιουργήσαμε μερικά πρότυπα στιμιότυπα από κάθε κλάση με συνολικό αριθμό τα εικοσιτέσσερα (24).

Επίλογος

Σε αυτό το σημείο έχουμε δημιουργήσει το Project στο NetBeans με τις απαραίτητες βιβλιοθήκες που χρειαστήκαμε και ορίσαμε ποιες κλάσεις θέλουμε να εισαχθούν στην βάση δεδομένων με τα κατάλληλα annotations. Συμπληρώσαμε το persistence.xml με αυτές τις κλάσεις και δημιουργήσαμε την σύνδεση με την βάση δεδομένων. Τελικό βήμα αυτής της διαδικασίας ήταν η εισαγωγή αντικειμένων στην βάση.

Πλέον έχουμε μία ολοκληρωμένη βάση δεδομένων την οποία είμαστε έτοιμοι για να χειριστούμε όπως εμείς επιθυμούμε, να ανακτήσουμε τα αντικείμενα που έχουμε αποθηκεύσει και να προχωρήσουμε περαιτέρω στην υλοποίηση της εφαρμογής. Επόμενο βήμα είναι η ανάκτηση πληροφοριών που θα αναλύσουμε στο κεφάλαιο 2.

ΚΕΦΑΛΑΙΟ 2 – Ανάκτηση Δεδομένων από την ObjectDB

ΕΙΣΑΓΩΓΗ

Για να είμαστε σε θέση να χειριστούμε τα αντικείμενα που έχουμε αποθηκευμένα στην βάση δεδομένων πρέπει να έχουμε πρόσβαση σε αυτήν. Το JPA API έχει έτοιμες μεθόδους για αυτόν τον λόγο χρησιμοποιώντας μάλιστα ψευδο-SQL ώστε να είμαστε εξοικειωμένοι με τις εντολές πριν καν εμβαθύνουμε σε αυτές.

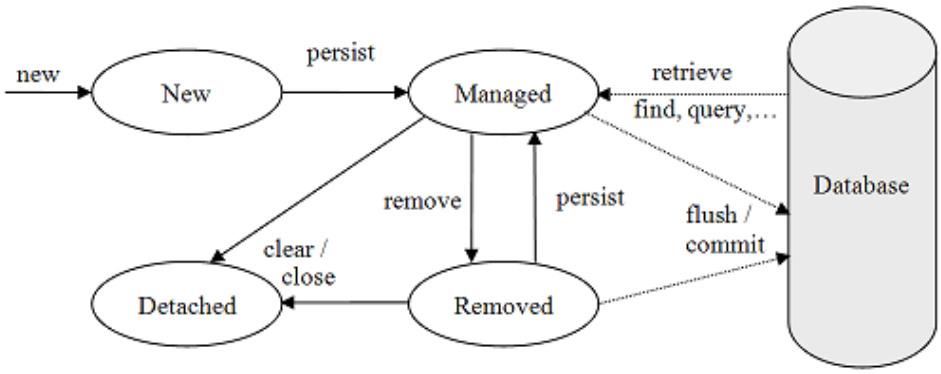
Σε αυτό το κεφάλαιο θα εξετάσουμε τις μεθόδους αυτές αφού ρίξουμε πρώτα μία ματιά στο Persistence Context που κάνει αυτές τις λειτουργίες ευκολότερες για εμάς.

2.1 Peristence Context

Το Java Persistence API (JPA) παρέχει διάφορους τρόπους ανάκτησης δεδομένων από την βάση δεδομένων. Η ανάκτηση τους δεν απαιτεί ξεχωριστή συναλλαγή (Transaction) από τον EntityManager καθώς δεν επηρεάζει το περιεχόμενο της βάσης. Τα Entity αντικείμενα είναι στιγμιότυπα αποθηκευμένα στην μνήμη, τα οποία αναπαριστούν φυσικά αντικείμενα που βρίσκονται στην βάση δεδομένων. Το JPA χρησιμοποιεί τον όρο Persistence Context για να περιγράψει την μέθοδο με την οποία συλλέγει όλα τα managed αντικείμενα από τον EntityManager. Αν κάποιο Entity αντικείμενο που θέλουμε να ανακτήσουμε υπάρχει ήδη στο Persistence Context, τότε αυτό επιστρέφεται χωρίς κάποια κλήση στην βάση δεδομένων.

Ο βασικός του ρόλος είναι να εξασφαλίσει ότι ένα αντικείμενο δεν αναπαρίσταται με πάνω από ένα στιγμιότυπα αποθηκευμένα στην μνήμη στον ίδιο EntityManager. Επομένως λειτουργεί ως προσωρινή μνήμη για την ανάκτηση αντικειμένων από την βάση δεδομένων. Στην περίπτωση που κάποιο αντικείμενο δεν υπάρχει στην προσωρινή του μνήμη τότε γίνειται κλήση στην βάση και αφού ανακτηθεί αποθηκεύεται και αυτό στο Peristence Context.

Για να κατανοήσουμε καλύτερα τον κύκλο ζωής των Entity Αντικειμένων αξίζει να μελετήσουμε το παρακάτω διάγραμμα



Σχήμα 7

Όπως είπαμε όταν δημιουργείται ένα αντικείμενο έχει την κατάσταση New και δεν έχει ακόμα σχετιστεί με κάποιον EntityManager. Αυτό γίνεται Managed όταν το «χειριστούμε» ως Entity μέσω της μεθόδου persist. Entity αντικείμενα που ανακτούμε από την βάση είναι επίσης στην κατάσταση Managed. Όταν τροποποιήσουμε ένα αντικείμενο μέσα σε ένα ενεργό Transaction, η αλλαγή

εντοπίζεται από τον EntityManager και γίνεται η ενημέρωση της βάσης (όταν γίνει commit).

Ένα αντικείμενο μπορεί επίσης να αλλάξει κατάσταση κατά την ανάκτηση του σε Removed χρησιμοποιώντας την μέθοδο remove. Σε αυτήν την περίπτωση διαγράφεται και από το Persistence Context αλλά και από την φυσική μνήμη. Τέλος υπάρχει και η κατάσταση Detached η οποία περιγράφει αντικείμενα που δεν έχουν διαγραφεί αλλά δεν υπάρχει κάποια ενεργή σύνδεση που να οδηγεί σε αυτά.

2.2 Τρόποι Ανάκτησης Αντικειμένων από την ObjectDB

Υπάρχουν διάφοροι τρόποι ανάκτησης αντικειμένων από την βάση δεδομένων μέσω του JPA API.

- Ανάκτηση μέσω κλάσης και πρωτεύοντος κλειδιού

  Κάθε Entity αντικείμενο μπορεί να αναγνωριστεί αν γνωρίζουμε την κλάση και το πρωτεύον κλειδί του. Ο EntityManager διαθέτει μία μέθοδο, την *find(Class c, int ID)* η οποία επιστρέφει αυτό ακριβώς το αντικείμενο που της ζητάμε.

- Ανάκτηση μέσω Eager Fetch

  Αυτός ο τρόπος μας δίνει την επίλογη της ανάκτησης μαζί με το ζητούμενο αντικείμενο και άλλων εξαρτώμενων από αυτό. Αν για παράδειγμα ένα αντικείμενο έχει ως πεδίο του κάποιο άλλο αντικείμενο, μπορούμε να χρησιμοποιήσουμε το fetchType.LAZY για να αποφύγουμε την ανάκτηση και του δεύτερου αντικειμένου

- Ανάκτηση μέσω Πλοήγησης και Πρόσβασης (Navigation and Access)

  Όποιο αντικείμενο συναντήσουμε μπορεί να ανακτηθεί ελεύθερα ανεξάρτητα από το fetch policy

- Ανάκτηση μέσω Ερωτημάτων (Query)

  Αυτή είναι η πιο ευέλικτη και διαδεδομένη μέθοδος ανάκτησης αντικειμένων από την βάση. Η επίσημη γλώσσα ερωτημάτων της JPA είναι η JPQL και με αυτήν μπορούμε να συντάξουμε απλά και πολύπλοκα ερωτήματα ανάλογα με τις ανάγκες μας.

- Ανάκτηση μέσω Refresh

Όταν χρησιμοποιούμε την μέθοδο refresh ανανεώνονται τα επιλεγμένα αντικείμενα οπότε γίνεται εκ νέου ανάκτηση τους

Στην εφαρμογή μας επιλέξαμε τον πιο διαδεδομένο τρόπο της ανάκτησης αντικειμένων μέσω ερωτημάτων. Τα ερωτήματα αναπαριστώνται στην JPA από δύο Interfaces. Το Query και το TypedQuery. Το πρώτο είναι παλαιότερο και για πιο γενική χρήση ενώ το δεύτερο, που είναι επέκταση του Query, χρησιμοποιείται κυρίως όταν γνωρίζουμε τον τύπο του αντικειμένου που θέλουμε να ανακτήσουμε. Παρακάτω δίνονται δύο παραδείγματα χρήσης αυτών των Interface.

```
TypedQuery<Object> query = em.createQuery("SELECT o FROM Object o",
Object.class);
```

```
Query query = em.createQuery("SELECT o FROM Object o");
```

Όπως βλέπουμε το Query παίρνει ως παράμετρο μόνο ένα String που περιέχει το ερώτημα που θέλουμε να εκτελέσουμε. Αντίθετα στο TypedQuery πρέπει να ορίσουμε και μία δεύτερη παράμετρο, τον τύπο των αναμενόμενων αποτελεσμάτων. Βέβαια στα συγκεκριμένα παραδείγματα το αποτέλεσμα είναι το ίδιο καθώς ως ζητούμενη κλάση ορίσαμε την Object οπότε θα μας επιστραφούν όλα τα αντικείμενα. Αν αντικαταστήσουμε την Object με την Athlete τότε θα ανακτήσουμε μόνο τα αντικείμενα τύπου Athlete.

Επιλέξαμε να χρησιμοποιήσουμε την TypedQuery παιρνόντας σαν παράμετρο την Object.class για την καλύτερη οργάνωση του κώδικα και για να μην επαναλαμβάνονται παρόμοια κομμάτια κώδικα με συνεχόμενες κλήσεις στην βάση για κάθε τύπο αντικειμένου (Athlete, Trainer, Stadium κτλ).

Η String παράμετρος που δέχεται η μέθοδος createQuery έχει σύνταξη παρόμοια της SQL πράγμα που μας διευκολύνει πολύ στην χρησιμοποίηση της.

Μετά από την εκτέλεση αυτής της εντολής, και ενώ φυσικά υπάρχει ήδη ανοικτή σύνδεση στην βάση δεδομένων, έχουμε στο collection query όλα τα αντικείμενα που υπάρχουν στην βάση δεδομένων. Ο ενδεδειγμένος τρόπος για να τα προσπελάσουμε είναι να τα αποθηκεύσουμε όλα σε μία λίστα αντικειμένων. Για να το επιτύχουμε αυτό χρησιμοποιούμε την μέθοδο getResultList()

```
List<Object> results = query.getResultList();
```

Επόμενο βήμα είναι η υλοποίηση ενός βρόγχου for με τον οποίο θα προσπελάσουμε όλα τα αντικείμενα (Τύπου Object προς το παρόν) και με την βοήθεια της instanceof θα καθορίσουμε τον τύπο τους και θα τα κάνουμε ώστε να μπορούμε να τα χρησιμοποιήσουμε.

```
for(Object result : results)
{
   if(result instanceof Athlete)
   {
      Athlete a = (Athlete)result;
   }
   else if(result instanceof Game)
   {
      Game g = (Game)result;
   }
   else if(result instanceof Helps)
   {
      Helps h = (Helps)result;
   }
   else if(result instanceof Judge)
   {
      Judge j = (Judge)result;
   }
   else if(result instanceof Participates)
   {
      Participates p = (Participates)result;
   }
   else if(result instanceof Sport)
   {
      Sport s = (Sport)result;
   }
   else if(result instanceof Stadium)
   {
      Stadium st = (Stadium)result;
   }
   else if(result instanceof Trainer)
   {
      Trainer t = (Trainer)result;
   }
   else if(result instanceof Volunteer)
   {
      Volunteer v = (Volunteer)result;
   }
}
```

Σε αυτόν τον βρόγχο βλέπουμε πως μπορούμε να έχουμε πρόσβαση σε κάθε αντικείμενο αλλά παραλήφθηκαν οι εντολές με τις οποίες ανακτήσαμε και τα

επιμέρους δεδομένα κάθε αντικειμένου καθώς θα παρουσιαστούν στο επόμενο κεφάλαιο.

2.3 Interface Metamodel

Σε αυτήν την πτυχιακή η βάση δεδομένων είναι γνωστή και οι δηλώσεις των κλάσεων υλοποιημένες από το Project. Σε πολλές περιπτώσεις όμως το μόνο που θα έχουμε είναι ένα αρχείο .odb (βάση δεδομένων της ObjectDB). Το να υλοποιήσουμε μία εφαρμογή που θα μπορεί να αναγνωρίζει και να επεξεργάζεται άγνωστους τύπους δεδομένων θα ήταν οπωσδήποτε πρόκληση, αλλά ξεπερνάει κατά πολύ τον σκοπό της πτυχιακής και ουσιαστικά αλλάζει το πεδίο ορισμού της. Παρακάτω θα κάνουμε μία μικρή αναφορά στο Interface Metamodel και στις δυνατότητες που μας παρέχει να εξερευνήσουμε άγνωστους τύπους δεδομένων. Σε συνδυασμό με το Reflection API της Java δημιουργεί ένα όχημα, όπου το άγνωστο γίνεται γνωστό και το αδύνατο δυνατό.

Το Interface Metamodel παρέχει διάφορες μεθόδους εξερεύνησης persistable types ορισμένων από τον χρήστη (managed). Αρχικά υπάρχουν τρεις μέθοδοι ανάκτησης set τυπών

```
// Get all the managed classes:
// (entity classes, embeddable classes, mapped super classes)
Set<ManagedType> allManagedTypes = metamodel.getManagedTypes();

// Get all the entity classes:
Set<EntityType> allEntityTypes = metamodel.getEntities();

// Get all the embeddable classes:
Set<EmbeddableType> allEmbeddableTypes = metamodel.getEmbeddables();
```

Αν κάποιες managed κλάσεις δεν είναι δηλωμένες στο persistence.xml τότε επιστρέφονται μόνο οι γνωστοί τύποι. Δηλαδή όλοι οι τύποι που περιέχουν στιγμιότυπα αποθηκευμένα στην βάση δεδομένων. Υπάρχουν μέθοδοι που επιστρέφουν κάποιους συγκεκριμένους τύπους.

Όταν η δήλωση μία κλάσης απουσιάζει η ObjectDB αυτόματα δημιουργεί μία συνθετική κλάση σύμφωνα με το σχήμα που έχει αποθηκευμένο στην βάση

δεδομένων. Αυτά τα δεδομένα μπορούμε να τα προσπελάσουμε χρησιμοποιώντας το Java Reflection API σε συνδυασμό με τις μεθόδους που μας παρέχει το JPA Metamodel API.

2.4 Συλλογή αποτελεσμάτων

Έχουμε δει μέχρι τώρα πως μπορούμε να δημιουργήσουμε και να ανακτήσουμε τα αντικείμενα από μία βάση δεδομένων. Για την καλύτερη λειτουργία της εφαρμογής δώσαμε την δυνατότητα σε κάθε αντικείμενα να περιγράφει τον εαυτό του μέσω της υπέρβασης της μεθόδου toString(). Παρακάτω θα δούμε πως συντάχθηκε αυτή η μέθοδος για το αντικείμενο Athlete.

Σκοπός μας είναι όλα τα αποτελέσματα να μορφοποιηθούν και να αποθηκευτούν σε ένα αρχείο RDF turtle.

```
   @Override
   public String toString()
   {
      String tmpAthlete = "";
      tmpAthlete += "<Athlete:" + this.Code +">\n";
      tmpAthlete += "\t:Name \"" + this.Name  +"\" ;\n";
      tmpAthlete += "\t:Surnname \"" + this.Surname  +"\" ;\n";
      tmpAthlete += "\t:Genre \"" + this.Genre  +"\" ;\n";
      tmpAthlete += "\t:DateOfBirth \"" + this.DateOfBirth  +"\" ;\n";
      tmpAthlete += "\t:Weight " + this.Weight  +" ;\n";
      tmpAthlete += "\t:Height " + this.Height  +" ;\n";
      tmpAthlete += "\t:CountryOfOrigin \"" + this.CountryOfOrigin  +"\"
;\n";
      tmpAthlete += "\t:CountryOfParticipation \"" +
this.CountryOfParticipation  +"\" .\n";
      return tmpAthlete;
   }
```

Με αυτόν τον τρόπο η toString μας εμφανίζει όταν καλείται την παρακάτω μορφοποίηση:

<Athlete:1>

:Name "Kwnstantinos" ;

:Surnname "Kenteris" ;

```
:Genre "Male" ;

:DateOfBirth "1970-01-01" ;

:Weight 84.3 ;

:Height 182 ;

:CountryOfOrigin "Greece" ;

:CountryOfParticipation "Greece" .
```

Το οποίο είναι αποδεκτός κόμβος εγγράφου Turtle που περιγράφει έναν Αθλητή με ID = 1, όνομα Kwnstantinos κτλ. Κάθε ιδιότητα του αθλητή χωρίζεται με το σημείο ; και η τελεία δηλώνει το τέλος της περιγραφής αυτού του αθλητή. Από εκεί και κάτω ότι γράψουμε δεν θα αναφέρεται σε αυτόν τον αθλητή.

Επίλογος

Η αποθήκευση δεδομένων σε μία βάση θα ήταν εντελώς άσκοπη αν δεν μπορούσαμε να τα ανακτήσουμε, να τα επεξεργαστούμε ή ακόμα και να τα διαγράψουμε. Σε αυτό κεφάλαιο παρουσιάσαμε πως υλοποιούνται αυτές οι λειτουργίες με μερικά ενδεικτικά κομμάτια κώδικα και παραδείγματα.

Στο επόμενο κεφάλαιο θα μελετήσουμε πως πρέπει να είναι δομημένο ένα έγγραφο RDF Turtle, που είναι και το ζητούμενο της πτυχιακής εργασίας, και πως έγινε η επεξεργασία των δεδομένων ώστε να ταιριάζει σε αυτό το πρότυπο.

ΚΕΦΑΛΑΙΟ 3 – Μετατροπή αποτελεσμάτων σε RDF Turtle έγγραφο

ΕΙΣΑΓΩΓΗ

Η Turtle (Terse RDF Triple Language) είναι ένα πρότυπο μορφοποίησης και παρουσίασης δεδομένων για το RDF (Resource Description Framework) μοντέλο δεδομένων με σύνταξη παρόμοια της SPARQL. Η RDF αναπαριστά την γνώση σε μορφή τριπλετών κάθε μία από τις οποίες αποτελείται από ένα υποκείμενο, μία ιδιότητα και ένα αντικείμενο. Η Turtle παρέχει μία μεθοδολογία για

να συνδιάζουμε τρία URIs ώστε να φτιάξουμε μία τριάδα και προσφέρει τρόπους συντόμευσης των προτάσεων όταν υπάρχουν πολλαπλές ιδιότητες για το ίδιο υποκείμενο. Για παράδειγμα αν θέλουμε να ορίσουμε έναν άνθρωπο ο οποίος έχει στην κατοχή του δύο αριθμούς τηλεφώνου δεν χρειάζεται να κάνουμε ξεχωριστή δήλωση για κάθε τηλέφωνο, αλλά τα συμπεριλαμβάνουμε στην ίδια.

Η RDF χρησιμοποιεί URIs επομένως η συγγραφή πολλών προτάσεων δυσκολεύει ειδικά αν γίνεται από άνθρωπο και όχι αυτόματα από τον υπολογιστή. Για αυτόν τον λόγο έχουν προβλεφθεί τα λεγόμενα prefixes. Στην αρχή κάθε εγγράφου δηλώνουμε τα URIs που θα επαναλαμβάνονται με συντομεύσεις και κάθε φορά που θέλουμε να χρησιμοποιήσουμε κάποιο, γράφουμε απλά την συντόμευση του.

Σε αυτό το κεφάλαιο θα εξηγήσουμε την σύνταξη των εγγράφων Turtle καθώς και τον τρόπο με τον οποίο μορφοποιήσαμε τα δεδομένα που ανακτήσαμε από την βάση δεδομένων βάση του αντίστοιχου προτύπου.

3.1 RDF Turlte

Υπάρχουν δύο τύποι δεδομένων στην RDF. Τα URIs και τα Literals. Τα URIs είναι αναφορές κάποιας οντότητας. Είναι συνήθως γραμμένα ως διευθύνσεις στο διαδίκτυο αλλά αυτό δεν είναι δεσμευτικό. Πρέπει όμως να περικλείονται σε <> αγκύλες. Αυτές ο αγκύλες δεν αναπαριστούν XML Tags, ούτε χρειάζεται να τα κλείσουμε με την κάθετο / (όπως στο
). Υπάρχουν απλά για να δηλώσουν ότι περικλείουν ένα URI μέσα τους.

Τα Literals είναι τύποι δεδομένων όπως String, Integer, Date κτλ. Όλοι οι τύποι δεδομένων πρέπει να περικλείονται σε quotes “” εκτός από τους καθαρά αριθμητικούς τύπους. Τα Literals μπορούν να συνταχθούν ως XSD τύποι χωρίς όμως να είναι απαραίτητο.

Οι τριπλέτες (triples) είναι ατομικά για κάθε οντότητα μπλοκ που αναπαριστούνται ως υποκείμενο – ιδιότητα – αντικείμενο. Το υποκείμενο και η ιδιότητα πρέπει να είναι URIs ενώ το αντικείμενο μπορεί να είναι είτε URI είτε Literal.

Γιατί όμως ο Σημασιολογικός Ιστός επιλέγει την RDF σύνταξη και όχι την ήδη υπάρχουσα και διαπρέπουσα XML; Το μοντέλο της XML μας επιτρέπει να αναπαριστούμε δεδομένα και όχι την σημασία των δεδομένων. Δεν μπορεί δηλαδή να αντιληφθεί την σχέση που έχουν τα δεδομένα. Η XML είναι κατά βάση ένα πρότυπο Serialization, εξειδικευμένο ώστε να περνάει πληροφορίες από μηχανή σε μηχανή. Αντίθετα το RDF είναι ένα μοντέλο δεδομένων με ένα αφηρημένο σύνολο κανόνων παρουσίασης πληροφορίας.

Κανόνες σύνταξης Turtle αρχείων:

- Κάθε τριπλέτα ξεκινάει σε μία νέα γραμμή και τελειώνει με την τελεία.
- Κενές γραμμές χρησιμοποιούνται για να χωρίσουν δύο αντικείμενα
- Τα σχόλια ξεκινούν με το σύμβολο # και ένα κενό και συνεχίζουν ως το τέλος της γραμμής
- Τα Literals γράφονται μέσα σε quotes “”
- Τα URIs περικλείονται σε αγκύλες <>
- Όταν συμπληρώνουμε ένα υποκείμενο, οι χαρακτήρες που προηγούνται των αγκυλών είναι συντομεύσεις που ορίζονται στην αρχή του εγγράφου, τα prefixes
- Υπάρχει η σύμβαση ότι όταν μετά από το κλείσιμο των αγκυλών ακολουθεί κεφαλαίο γράμμα, η επόμενη λέξη αντιπροσωπεύει μία κλάση ή αλλιώς μία οντότητα.
- Το γράμμα “a” όταν χρησιμοποιείται ως ιδιότητα είναι συντόμευση για το “rdf:type” και σημαίνει ότι “είναι τύπου”

Ένα ολοκληρωμένο αρχείο Turtle μπορεί να έχει την παρακάτω μορφή:

```
@prefix Athlete: http://it.teithe.gr/Rdf/Turtle/Athlete/
<Athlete:1>
        :Name "Kwnstantinos" ;
        :Surnname "Kenteris" ;
        :Genre "Male" ;
        :DateOfBirth "1970-01-01" ;
        :Weight 84.3 ;
        :Height 182 ;
        :CountryOfOrigin "Greece" ;
        :CountryOfParticipation "Greece" .
```

3.2 Μορφοποίηση δεδομένων σε Turtle μορφή

Όπως είπαμε και προηγουμένως έχουμε υπερβεί την μέθοδο των αντικειμένων toString ώστε να μας επιστρέφει τα δεδομένα του κάθε αντικειμένου σε αποδεκτή Turtle μορφή. Για χάρην ευκολίας ανάγνωσης των αποτελεσμάτων προτιμήσαμε τα URIs να έχουν την μορφή απλών λέξεων και όχι ιστοσελίδων με εξαίρεση τον ορισμό των υποκειμένων. Σε αυτά θεωρήσαμε ότι βρίσκονται όλα σε μία εικονική σελίδα, την http://it.teithe.gr/Rdf/Turtle/

Η μέθοδος toString έχει την παρακάτω μορφή για το αντικείμενο Athlete

```
    @Override
    public String toString()
    {
        String tmpAthlete = "";
        tmpAthlete += "<Athlete:" + this.Code +">\n";
        tmpAthlete += "\t:Name \"" + this.Name  +"\" ;\n";
        tmpAthlete += "\t:Surnname \"" + this.Surname  +"\" ;\n";
        tmpAthlete += "\t:Genre \"" + this.Genre  +"\" ;\n";
        tmpAthlete += "\t:DateOfBirth \"" + this.DateOfBirth  +"\" ;\n";
        tmpAthlete += "\t:Weight " + this.Weight  +" ;\n";
        tmpAthlete += "\t:Height " + this.Height  +" ;\n";
        tmpAthlete += "\t:CountryOfOrigin \"" + this.CountryOfOrigin  +"\"
;\n";
        tmpAthlete += "\t:CountryOfParticipation \"" +
this.CountryOfParticipation  + " ;\n";
        tmpAthlete += "\t:Trainer " + this.trainer.Code  + " .\n";
        return tmpAthlete;
    }
```

Αυτό όμως συμπεριλαμβάνει μόνο την οντότητα Athlete και δεν είναι ένα ολοκληρωμένο Turtle έγγραφο. Για να έχει νόημα η παραπάνω μορφή πρέπει να έχουμε ορίσει το prefix “Athlete:”. Για αυτόν τον λόγο δημιουργήσαμε μία μέθοδο με όνομα GetPrefixes().

```
    private String GetPrefixes()
    {
        String strPrefixes = "";
        try
        {
            OpenConnections();
            Metamodel metamodel = em.getMetamodel();
            Set<ManagedType<?>> allManagedTypes=metamodel.getManagedTypes();
            String ittei = "http://it.teithe.gr/Rdf/Turtle/";

            for(ManagedType m : allManagedTypes)
            {
                String strClass = m.getJavaType().getName();
                strPrefixes += "@prefix "+strClass+": "+ittei+strClass+"/\n";
            }
        } catch(Exception ex)
        {
            strMyTtlFile = "An exception occured...\n"+ex.getMessage()+"\n";
            strMyTtlFile += "\nStack Trace:\n" + ex.getStackTrace();
```

```
            System.out.println(ex.getMessage());
            System.out.println(ex.getStackTrace());
        }
        return strPrefixes;
    }
```

Με την βοήθεια του Metamodel API βρίσκουμε όλους τους managed τύπους που είναι αποθηκευμένοι στην βάση δεδομένων και έπειτα με ένα βρόγχο for δημιουργούμε τα URIs για κάθε τύπο αντικειμένου και τα αποθηκεύουμε σε ένα String. Μέσω της εντολής getJavaType().getName() παίρνουμε το όνομα της κλάσης ως String. Στο String strPrefixes προσθέτουμε κάθε ένα από τα prefixes και στην συνέχεια αυτό το String θα είναι το πρώτο που θα γραφτεί στο αρχείο .ttl που θα δημιουργήσουμε.

Επόμενο βήμα είναι να ανακτήσουμε όλα τα αντικείμενα από την βάση, το οποίο είδαμε πιο πάνω πως το υλοποιήσαμε. Με την βοήθεια της TypedQuery πήραμε τα δεδομένα και με την instanceof κάναμε cast κάθε τύπο αντικειμένου για να μπορέσουμε να χρησιμοποιήσουμε την toString μέθοδο του. Να σημειώσουμε ότι χρησιμοποιήσαμε εννέα προσωρινά Strings ώστε τα αποτελέσματα μας να είναι ταξινομημένα, καθώς η ObjectDB δεν μας τα επιστρέφει στην σειρά που τα εισάγαμε.

Με την μέθοδο WriteDataToFile(String str) ανοίγουμε το αρχείο «OlympicGames.ttl» και εισάγουμε σε αυτό τα δεδομένα μας. Αν δεν υπάρχει το αρχείο δημιουργείται ένα καινούριο.

```
    private boolean WriteDataToFile(String strToWrite)
    {
        try
        {
            PrintWriter writer=new PrintWriter("OlympicGames.ttl", "UTF-8");
            writer.println(strToWrite);
            writer.close();
            txtInformation.setText(strToWrite.equals("")
                                   ? "Empty file created because there are
no data in the database..."
                                   : "File creation was successfull!");
        }
        catch(Exception ex)
        {
            strMyTtlFile="An exception occured...\n"+ex.getMessage() + "\n";
            strMyTtlFile += "\nStack Trace:\n" + ex.getStackTrace();
            System.out.println(ex.getMessage());
```

```
            System.out.println(ex.getStackTrace());
            return false;
        }
        return true;
    }
```

Στην εντολή setText κάνουμε έναν απλό έλεγχο με τον τριαδικό τελεστή της Java. Αν το String που αποθηκεύουμε τα αποτελέσματα είναι κενό, στέλνουμε ένα αντίστοιχο μήνυμα αλλιώς ενημερώνουμε ότι όλα λειτούργησαν σωστά.

3.3 Έλεγχος Αποτελεσμάτων

Με την δημιουργία του αρχείου Turtle έχουμε ολοκληρώσει τον στόχο της πτυχιακής εργασίας. Πλέον έχουμε μία δομή δεδομένων έτοιμη να χρησιμοποιηθεί στον Σημασιολογικό Ιστό και τα Διασυνδεδεμένα Δεδομένα. Παρακάτω παρουσιάζεται τμήμα του αρχείου για τον προληπτικό έλεγχο των αποτελεσμάτων.

OlympicGames.ttl

```
@prefix Trainer: http://it.teithe.gr/Rdf/Turtle/Trainer/
@prefix Athlete: http://it.teithe.gr/Rdf/Turtle/Athlete/
@prefix Sport: http://it.teithe.gr/Rdf/Turtle/Sport/
@prefix Judge: http://it.teithe.gr/Rdf/Turtle/Judge/
@prefix Stadium: http://it.teithe.gr/Rdf/Turtle/Stadium/
@prefix Game: http://it.teithe.gr/Rdf/Turtle/Game/
@prefix Volunteer: http://it.teithe.gr/Rdf/Turtle/Volunteer/
@prefix Helps: http://it.teithe.gr/Rdf/Turtle/Helps/
@prefix Participates: http://it.teithe.gr/Rdf/Turtle/Participates/
<Athlete:1>
        :Name "Kwnstantinos" ;
        :Surnname "Kenteris" ;
        :Genre "Male" ;
        :DateOfBirth "1970-01-01" ;
        :Weight 84.3 ;
        :Height 182 ;
        :CountryOfOrigin "Greece" ;
        :CountryOfParticipation "Greece ;
        :Trainer 1 .
<Trainer:1>
        :Name "Christos" ;
        :Surnname "Tzekos" ;
        :Genre "Male" ;
        :CountryOfOrigin "Greece" .
<Sport:1>
```

```
        :Name "100m Men" ;
        :Genre "Male" ;
        :DateOfRecord "1970-01-01" ;
        :Record 9.39 ;
        :AthleteCode 4 .
<Game:1>
        :GameLevel "World" ;
        :DateOfGame "1970-01-01" ;
        :TimeOfGame "02:00:01" ;
        :StadiumCode 1 ;
        :SportCode 1 ;
        :Judge 1 .
<Participates:1>
        :GameCode 1 ;
        :AthleteCode 1 ;
        :Performance 9.76 ;
        :Valid "true" .
<Stadium:1>
        :Name "Marakana" ;
        :Capacity "120000" ;
        :City "Rio De Janeiro" .
<Judge:1>
        :Name "Petros" ;
        :Surnname "Petridis" ;
        :Genre "Male" ;
        :DateOfBirth "1970-01-01" .
<Volunteer:1>
        :Name "Giorgos" ;
        :Surnname "Georgiadis" ;
        :Genre "Male" ;
        :Age "23" ;
        :AT "A313136" .
<Helps:1>
        :GameCode 1 ;
        :VolunteerCode 1 .
```

Από κάθε οντότητα κρατήσαμε ένα στιγμιότυπο για λόγους συντομίας. Στην αρχή του εγγράφου περιγράφονται τα prefixes, ένα για κάθε οντότητα. Έπειτα δηλώνουμε κάθε στιγμιότυπο ξεχωριστά, χωρισμένο με μία τελεία, ξεκινώντας από το URI του μέσα σε αγκύλες και στην συνέχεια δηλώνουμε την κάθε ιδιότητα με την τιμή της.

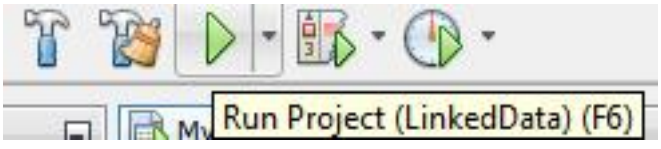
ΕΠΙΛΟΓΟΣ

Σε αυτό το κεφάλαιο μελετήσαμε πως λειτουργεί η RDF Turtle σύνταξη, σε τι αποσκοπεί και μέσω κώδικα Java μετατρέψαμε τα αντικείμενα που είχαμε αποθηκευμένα στην βάση δεδομένων σε μορφή Turtle ώστε να εισαχθούν σε ένα αρχείο. Η δουλειά μας έχει ολοκληρωθεί. Επόμενο βήμα θα ήταν η εισαγωγή του αρχείου στο διαδίκτυο ως Linked Data. Αυτό θα ήταν δυνατό στην περίπτωση που τα URIs αντιπροσώπευαν πραγματικές ιστοσελίδες και τα δεδομένα ήταν ακριβή και αληθή και όχι δοκιμαστικά.

ΚΕΦΑΛΑΙΟ 4 – Οδηγός χρήσης της εφαρμογής

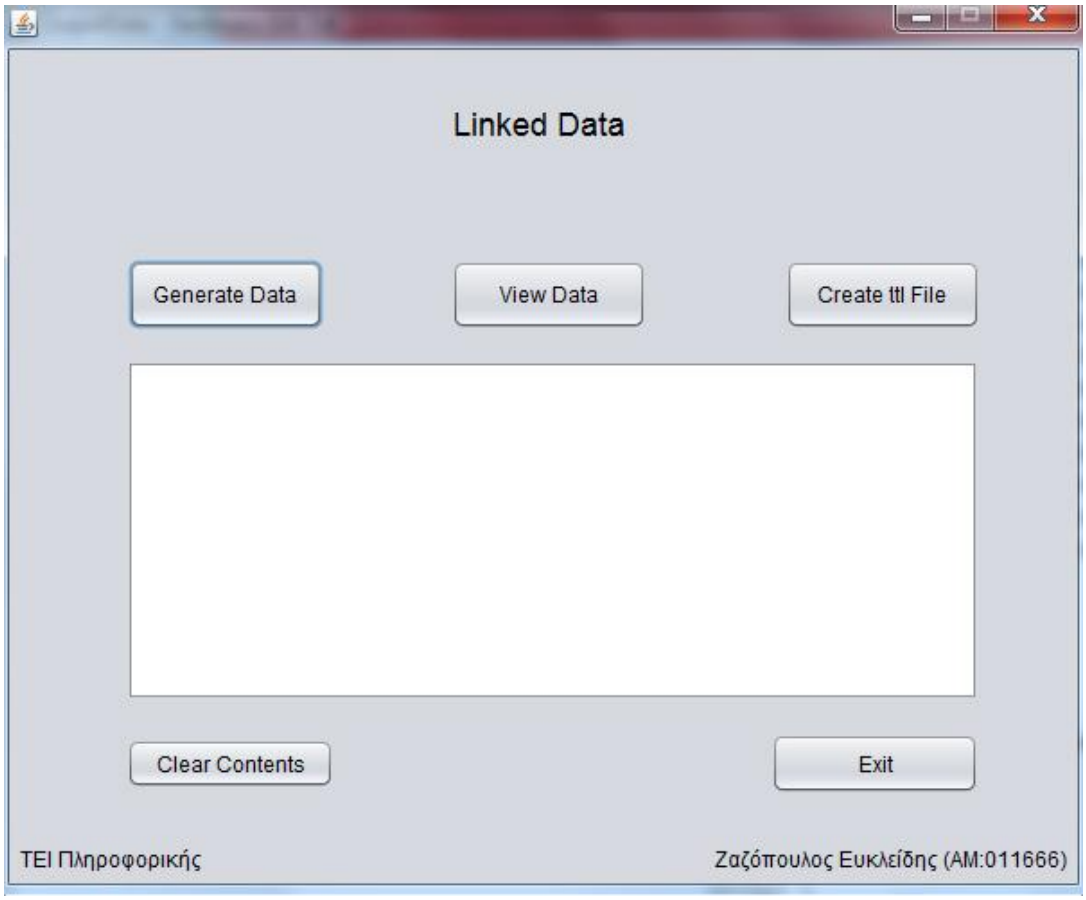
Ο στόχος της εφαρμογής που υλοποιήθηκε είναι να μετατρέπει αντικείμενα από μία αντικειμενοστρεφή βάση δεδομένων σε μορφή αποδεκτή από την RDF Turtle, ώστε να μπορούν να χρησιμοποιηθούν ως Linked Data στο διαδίκτυο. Πως όμως λειτουργεί η εφαρμογή; Μπορούμε να την εκκινήσουμε είτε κάνοντας compile το Project από το NetBeans, είτε εκτελώντας το .jar αρχείο που δημιουργεί αυτόματα το IDE κατά την μεταγλώττιση του κώδικα. Αν επιλέξουμε τον δεύτερο τρόπο πρέπει να έχουμε υπ' όψην μας ότι αν έχουμε κάνει κάποιες αλλαγές χωρίς να ακολουθήσει compile, δεν θα τις έχει συμπεριλάβει.

Ξεκινώντας λοιπόν από το IDE πατάμε το κουμπί Run Project ή την συντόμευση F6 από το πληκτρολόγιο.



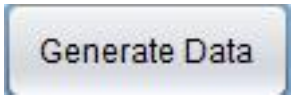
Σχήμα 8

Η εφαρμογή έχει ένα κεντρικό παράθυρο με πέντε κουμπιά και ένα Πεδίο Κειμένου όπως φαίνεται στην παρακάτω εικόνα.



Σχήμα 9

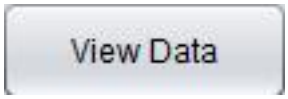
Παρακάτω εξηγούνται αναλυτικά οι λειτουργίες που εκτελεί κάθε κουμπί.



Σχήμα 10

Το κουμπί αυτό ανοίγει την σύνδεση με την βάση δεδομένων και με την βοήθεια της TypedQuery ανακτεί όλα τα αντικείμενα που είναι αποθηκευμένα σε αυτήν. Έπειτα ελέγχει τον αριθμό των εγγραφών που επεστράφησαν και αν είναι περισσότερες από μηδέν, θέτει την τιμή του textbox σε «Records allready exist!». Αλλιώς καλεί την μέθοδο FillWithData() η οποία είναι υπεύθυνη για την εισαγωγή των αντικειμένων στην βάση δεδομένων. Η μέθοδος είναι τύπου Boolean και επιστρέφει True αν όλα εκτελεστούν σωστά, ή False αν υπάρξει κάποιο σφάλμα. Αν εκτελεστεί χωρίς σφάλματα δίνει στο textbox την τιμή «Data generated successfully!». Αυτό γίνεται για να αποφύγουμε τις άσκοπες αναλύσεις και μετατροπές των αντικειμένων σε RDF κείμενο.

Μετά την εκτέλεση αυτού του κομματιού κώδικα δεν έχουμε δημιουργήσει το αρχείο που επιζητούμε αλλά τα δεδομένα με τα οποία θα το γεμίσουμε.



Σχήμα 11

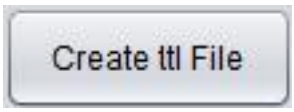
Επόμενη λειτουργία της εφαρμογής είναι να ελέγξουμε τα δεδομένα που δημιουργήθηκαν. Αυτό γίνεται πατώντας το κουμπί View Data. Η πρώτη εντολή που εκτελείται είναι ένας απλός έλεγχος αν η σύνδεση με την βάση είναι ανοιχτή. Αν δεν υπάρχει σύνδεση τότε δημιουργούμε ένα νέο EntityManager.

Μετά από αυτό το σημείο χρησιμοποιούμε δύο μεθόδους: την GetPrefixes() και την GetAllData(). Η GetPrefixes όπως καταλαβαίνουμε και από το όνομα της επιστρέφει ένα String με όλα τα prefixes από τους τύπους αντικειμένων που βρέθηκαν στην βάση δεδομένων. Η GetAllData διαβάζει όλα τα αντικείμενα και αποθηκεύει σε ένα String την μέθοδο toString του καθενός. Χρησιμοποιεί την TypedQuery και στην συνέχεια κάνοντας χρήση του instanceof Operator της Java αναγνωρίζει τον τύπο κάθε αντικείμενο (όπως είπαμε νωρίτερα η TypedQuery μας επιστρέφει όλα τα αντικείμενα αλλά τύπου Object).

```
TypedQuery<Object> query = em.createQuery("SELECT o FROM Object o",
Object.class);
List<Object> results = query.getResultList();


for(Object result : results)
{
    if(result instanceof Athlete)
    {
        Athlete a = (Athlete)result;
        strAthlete += a.toString();
    }
}
```

Επόμενο βήμα είναι η ένωση των String όλων των τύπων αντικειμένων σε ένα το οποίο επιστρέφεται από την μέθοδο GetAllData(). Έτσι έχουμε σε ένα String όλα τα δεδομένα που επιθυμούμε να εισάγουμε στο αρχείο Turtle.



Σχήμα 12

Το κουμπι Create ttl File κάνει μία πολύ απλή λειτουργία (αφού ελέγξει κι αυτό με την σειρά για την σύνδεση στην βάση δεδομένων): Ενώνει τα αποτελέσματα από τις δύο μεθόδους GetPrefixes και GetAllData και καλεί την WriteDataToFile(String str) η οποία δημιουργεί ένα νέο αρχείο ή ανανεώνει το ήδη υπάρχων με τα δεδομένα. Αν δεν βρει κανένα αποτέλεσμα και το String είναι κενό ενημερώνει στο textbox ότι «There are no data in the database...». Αλλιώς με το μήνυμα «File creation was successfull!» μας ενημερώνει ότι όλα κύλησαν σωστά.

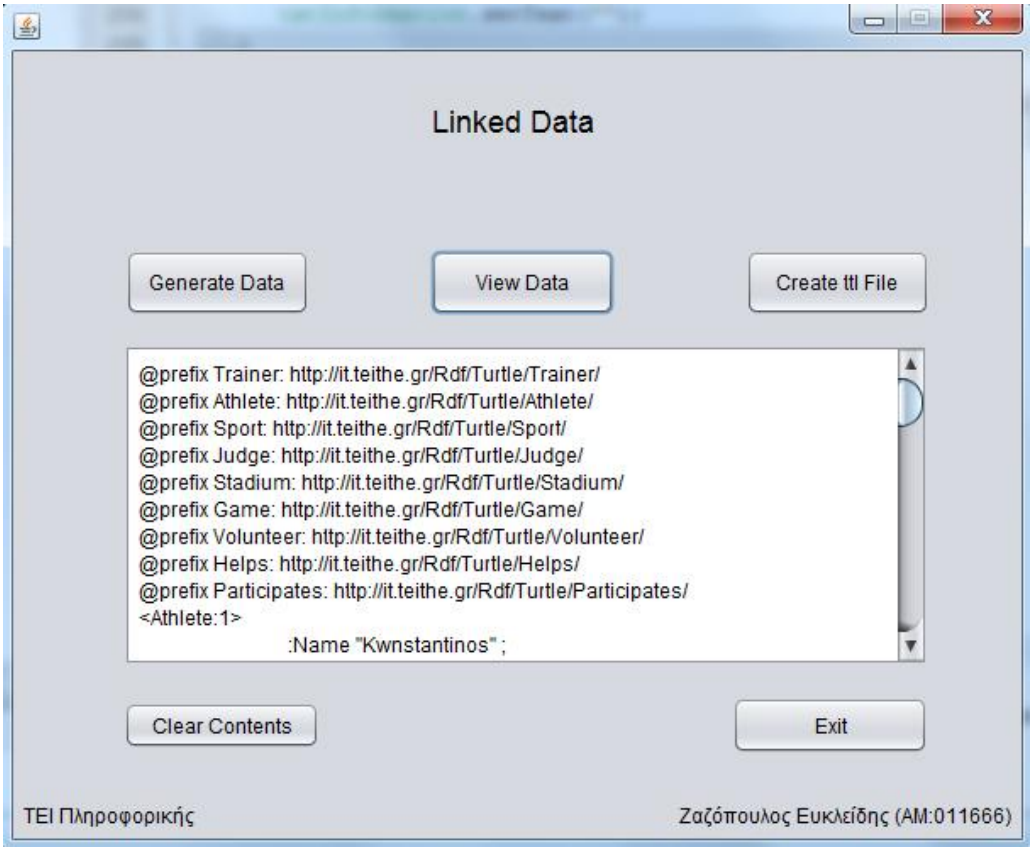
Σχήμα 13

Μετά από μερικές εκτελέσεις των παραπάνω λειτουργιών, δεν μπορούμε να είμαστε σίγουροι ότι τα δεδομένα μας ανανεώνονται ή αν απλά βρίσκονται στην μνήμη του συστήματος. Για αυτόν τον λόγο δημιουργήθηκε ένα ακόμα κουμπί, το Clear Contents με το οποίο καθαρίζουμε ότι δεδομένα είχαμε ανακτήσει ως τώρα.

Η πρώτη λειτουργία του είναι να κλείσει την σύνδεση με την βάση δεδομένων αν αυτή είναι ανοιχτή και να διαγράψει τα αρχεία που δημιουργεί η ObjectDB (το βασικό και το προσωρινό). Έπειτα καθαρίζει το textbox.

Σχήμα 14

Το κουμπι Exit όπως καταλαβαίνουμε χρησιμοποιείται για την έξοδο από την εφαρμογή. Αρχικά κλείνει όλα τα ανοιχτά connections, στην συνέχεια διαγράφει τα αρχεία που δημιούργησε η ObjectDB και τελικά κλείνει την εφαρμογή.



Σχήμα 15

Στο κέντρο της εφαρμογής υπάρχει ένα JTextArea με ένα JScrollPane υλοποιημένο. Αυτό χρησιμοποιείται για δύο λόγους. Πρώτον για να εμφανίζει τα αποτελέσματα πριν αποθηκευτούν στο αρχείο turtle και δεύτερον για να μας μεταφέρει σημαντικές για την εφαρμογή πληροφοριές σχετικά με την επιτυχία ή αποτυχία κάποιων λειτουργιών.

ΚΕΦΑΛΑΙΟ 5 - ΣΥΜΠΕΡΑΣΜΑΤΑ

Έχουμε υλοποιήσει μία εφαρμογή η οποία διαβάζει κάποιες πληροφορίες, τις μορφοποιεί και τις αποθηκεύει σε ένα αρχείο κειμένου. Τι είναι αυτές οι πληροφορίες όμως; Γιατί μας χρειάζονται και πως μπορούμε να τις χρησιμοποιήσουμε;

Τι είναι αυτό που βλέπουμε;

Στην τελευταία φάση της εφαρμογής έχουμε στα χέρια μας ένα αρχείο με κάποιες πληροφορίες όπως οι παρακάτω:

```
@prefix Trainer: http://it.teithe.gr/Rdf/Turtle/Trainer/
@prefix Athlete: http://it.teithe.gr/Rdf/Turtle/Athlete/
@prefix Sport: http://it.teithe.gr/Rdf/Turtle/Sport/
@prefix Judge: http://it.teithe.gr/Rdf/Turtle/Judge/
@prefix Stadium: http://it.teithe.gr/Rdf/Turtle/Stadium/
@prefix Game: http://it.teithe.gr/Rdf/Turtle/Game/
@prefix Volunteer: http://it.teithe.gr/Rdf/Turtle/Volunteer/
@prefix Helps: http://it.teithe.gr/Rdf/Turtle/Helps/
@prefix Participates: http://it.teithe.gr/Rdf/Turtle/Participates/
<Athlete:1>
        :Name "Kwnstantinos" ;
        :Surnname "Kenteris" ;
        :Genre "Male" ;
        :DateOfBirth "1970-01-01" ;
        :Weight 84.3 ;
        :Height 182 ;
        :CountryOfOrigin "Greece" ;
        :CountryOfParticipation "Greece ;
        :Trainer 1 .
<Trainer:1>
        :Name "Christos" ;
        :Surnname "Tzekos" ;
        :Genre "Male" ;
        :CountryOfOrigin "Greece" .
<Sport:1>
```

```
:Name "100m Men" ;
:Genre "Male" ;
:DateOfRecord "1970-01-01" ;
:Record 9.39 ;
:AthleteCode 4 .
```

……

Πρόκειται για ένα αρχείο RDF με την κωδικοποίηση της Turtle, που περιέχει ένα σύνολο από υποκείμενα τα οποία έχουν κάποιες ιδιότητες. Για παράδειγμα το υποκείμενο Athlete με κωδικό 1 έχει ως ιδιότητα το όνομα του που είναι «Kwnstantinos». Επίσης έχει έναν προπονητή (Trainer) με κωδικό 1. Η διαδύνδεση των δεδομένων είναι προφανής για τους ανθρώπους αλλά πιο σημαντικά για τους υπολογιστές, που πλέον μέσω του Σημασιολογικού Ιστού μπορούν να αντιληφθούν όχι μόνο απλά δεδομένα αλλά και σχέσεις μεταξύ τους, όπως ότι ο Αθλητής 1 έχει ως προπονητή τον 1 ο οποίος έχει όνομα «Christos» και ούτω καθεξής.

Είναι πολύ σημαντικό να κατανοήσουμε ότι αυτοί που αποσκοπούμε να επωφεληθούν από τέτοιου είδους συσχετίσεις και παρουσιάσεις πληροφοριών είναι οι υπολογιστές.

Πως μπορούν να χρησιμοποιηθούν τα αποτελέσματα;

Τα δεδομένα όπως τα παρουσιάσαμε παραπάνω είναι πρότυπα και δεν έχουν άμεση εφαρμογή. Για να γίνουν προσπελάσιμα πρέπει να τοποθετηθούν σε ιστοσελίδες με κατά προτίμηση γραφικό περιβάλλον και να υλοποιηθούν οι συσχετίσεις. Η πτυχιακή δεν έχει σκοπό να παρουσιάσει απτά αποτελέσματα αλλά να δείξει ότι δεδομένα αποθηκευμένα σε αντικειμενοστρεφείς βάσεις δεδομένων μπορούν να επεξεργαστούν και να μορφοποιηθούν κατάλληλα ώστε να συνδεθούν με άλλα ίδιου τύπου στο διαδίκτυο. Μέχρι τώρα το κυρίαρχο περιεχόμενο των Linked Data που υπάρχουν διαθέσιμα προέρχεται από το περιεχόμενο των ιστοσελίδων και από σχεσιακές βάσεις δεδομένων. Εδώ δείξαμε ότι αυτό είναι δυνατό και για δεδομένα που βρίσκονται σε OODBMS.

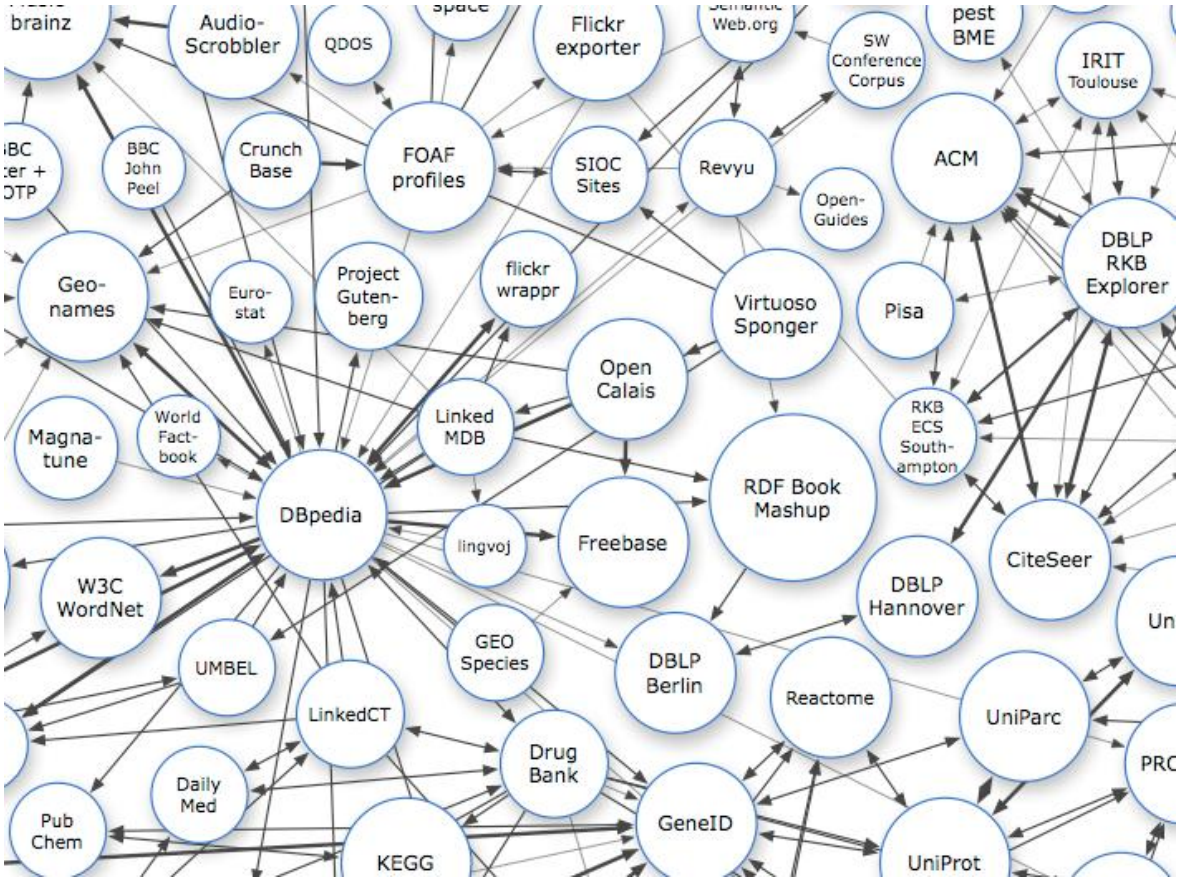
Έχει εφαρμογές κάτι τέτοιο στο διαδίκτυο;

Σήμερα που τα Διασυνδεδεμένα Δεδομένα αναπτύσσονται και εξελίσσονται ραγδαία κάθε πληροφορία μπορεί να ενσωματωθεί και να συνδεθεί με άλλες

δημιουργώντας έναν διαδικτυακό κόσμο γνώσης. Προσθέτοντας σε αυτόν τον κόσμο μία πληροφορία μπορεί να δημιουργήσει χιλιάδες συνδέσεις και να μας βοηθήσει στην ενημέρωση μας, την εκπαίδευση μας ακόμα και στην μόρφωση μας.

Τα Linked Data είναι μία επιπλέον ικανότητα που εμείς οι άνθρωποι αφομοιώνουμε σιγά σιγά καθημερινά. Την ικανότητα απόκτησης πληροφοριών και γνώσης. Μπορούμε να εξερευνούμε όλο και περισσότερα πράγματα και ο βαθμός με τον οποίο αυξάνεται αυτή η ικανότητα είναι εκθετικός.

Όπως είναι κατανοητό όταν τα δεδομένα αρχίζουν και συνδέονται μεταξύ τους βοηθούν τους προγραμματιστές να υλοποιήσουν ακόμα πιο ισχυρές εφαρμογές που θα εκμεταλλεύονται το διαδίκτυο και τους πόρους που αυτό προσφέρει στο έπακρον.



Σχήμα 16

Στο Σχήμα 16 βλέπουμε μέρος του Linking Open Data Project Cloud Diagram (http://lod-cloud.net/)

Πρόβλεψη / Εκτίμηση για το μέλλον των Linked Data

Το μέλλον του Διαδικτύου προκάλεσε το ενδιαφέρον πολλών διαφορετικών κοινοτήτων. Όλες αυτές οι κοινότητες αναπτύσουν συγκεκριμένα τμήματα υποδομής, η οποία κάποια στιγμή θα πρέπει να είναι σε θέση να λειτουργεί. Δυστυχώς, επί του παρόντος η μελλοντική αρχιτεκτονική του Διαδικτύου δεν περιλαμβάνει τα μέσα για την επίτευξη της διαλειτουργικότητας σε επίπεδο δεδομένων. Ταυτόχρονα τα Linked Data μετατρέπονται σε μία αποδεκτή βέλτιστη τεχνική για την ανταλλαγή πληροφοριών με έναν διαλειτουργικό και επαναχρησιμοποιήσιμο τρόπο. Πολλές διαφορετικές κοινότητες του Διαδικτύου χρησιμοποιούν τα Linked Data για να παρέχουν και ανταλλάσουν πληροφορίες. Αυτό επιβεβαιώνεται από το δραματικά αυξανόμενο Linked Data Cloud και τα - επί του παρόντος - περισσότερα από 25 δισεκατομμύρια γεγονότα που αντιπροσωπεύονται και διασυνδέονται εκεί με εκθετικό ρυθμό ανάπτυξης, τόσο από την άποψη των συνόλων δεδομένων όσο και των περιεχομένων τους.

Μια αποδεκτή περιγραφή των Linked Data ως ανεξάρτητο στρώμα στην αρχιτεκτονική του Διαδικτύου, είναι πάνω από το επίπεδο δικτύου, αλλά κάτω από τα επίπεδα των εφαρμογών, δεδομένου ότι παρέχει ένα κοινό μοντέλο δεδομένων για όλες τις εφαρμογές. Είναι το μέσο για την μετάβαση από το Διαδίκτυο των Δεδομένων που υπάρχει σήμερα στο Διαδίκτυο της γνώσης.

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Tim Berners-Lee, James Hendler, Ora Lassila, The Semantic Web, Scientific American, Μάιος 2001.

2. Κεραμόπουλος Ευκλείδης - Διαφάνειες Εργαστηρίων (Linked Data, RDF)

3. Μιχάλης Βαφόπουλος - Διαφάνειες Εργαστηρίων Linked Data: The Transformers

4. H. Schildt, 2007. Swing A Beginner's Guide, McGraw Hill

5. Tim Berners-Lee Date: 2006-07-27 - http://www.w3.org/DesignIssues/LinkedData.html

6. JPA 2 | Understanding relationships between entities - http://www.kumaranuj.com/2013/05/jpa-2-understanding-relationships.html

7. Getting Started with JPA and NetBeans - http://www.objectdb.com/tutorial/jpa/netbeans

8. http://linkeddata.org/

9. The Java EE 6 Tutorial - http://docs.oracle.com/javaee/6/tutorial/doc/bnbqa.html

10. Linked Open Vocabulary (LOV) http://lov.okfn.org/dataset/lov/index.html

ΠΑΡΑΡΤΗΜΑΤΑ

Ολοκληρωμένο αρχείο OlympicGames.ttl

```
@prefix Trainer: http://it.teithe.gr/Rdf/Turtle/Trainer/
@prefix Athlete: http://it.teithe.gr/Rdf/Turtle/Athlete/
@prefix Sport: http://it.teithe.gr/Rdf/Turtle/Sport/
@prefix Judge: http://it.teithe.gr/Rdf/Turtle/Judge/
@prefix Stadium: http://it.teithe.gr/Rdf/Turtle/Stadium/
@prefix Game: http://it.teithe.gr/Rdf/Turtle/Game/
@prefix Volunteer: http://it.teithe.gr/Rdf/Turtle/Volunteer/
@prefix Helps: http://it.teithe.gr/Rdf/Turtle/Helps/
@prefix Participates: http://it.teithe.gr/Rdf/Turtle/Participates/
<Athlete:1>
        :Name "Kwnstantinos" ;
        :Surnname "Kenteris" ;
        :Genre "Male" ;
        :DateOfBirth "1970-01-01" ;
        :Weight 84.3 ;
        :Height 182 ;
        :CountryOfOrigin "Greece" ;
        :CountryOfParticipation "Greece ;
        :Trainer 1 .
<Athlete:2>
        :Name "Katerina" ;
        :Surnname "Thanou" ;
        :Genre "Female" ;
        :DateOfBirth "1970-01-01" ;
        :Weight 54.3 ;
        :Height 167 ;
        :CountryOfOrigin "Greece" ;
        :CountryOfParticipation "Greece ;
        :Trainer 1 .
<Athlete:3>
        :Name "Konstantinos" ;
        :Surnname "Douvalidis" ;
        :Genre "Male" ;
        :DateOfBirth "1970-01-01" ;
        :Weight 78.0 ;
        :Height 184 ;
```

```
        :CountryOfOrigin "Greece" ;
        :CountryOfParticipation "Greece ;
        :Trainer 2 .
<Athlete:4>
        :Name "Usain" ;
        :Surnname "Bolt" ;
        :Genre "Male" ;
        :DateOfBirth "1970-01-01" ;
        :Weight 94.0 ;
        :Height 192 ;
        :CountryOfOrigin "Jamaica" ;
        :CountryOfParticipation "Jamaica ;
        :Trainer 3 .
<Trainer:1>
        :Name "Christos" ;
        :Surnname "Tzekos" ;
        :Genre "Male" ;
        :CountryOfOrigin "Greece" .
<Trainer:2>
        :Name "Mpampis" ;
        :Surnname "Sdrolas" ;
        :Genre "Male" ;
        :CountryOfOrigin "Greece" .
<Trainer:3>
        :Name "Glen" ;
        :Surnname "Mills" ;
        :Genre "Male" ;
        :CountryOfOrigin "Jamaica" .
<Sport:1>
        :Name "100m Men" ;
        :Genre "Male" ;
        :DateOfRecord "1970-01-01" ;
        :Record 9.39 ;
        :AthleteCode 4 .
<Sport:2>
        :Name "200m Men" ;
        :Genre "Male" ;
        :DateOfRecord "1970-01-01" ;
        :Record 19.21 ;
        :AthleteCode 4 .
<Sport:3>
        :Name "100m Women" ;
        :Genre "Female" ;
        :DateOfRecord "1970-01-01" ;
        :Record 10.15 ;
        :AthleteCode 2 .
<Game:1>
        :GameLevel "World" ;
        :DateOfGame "1970-01-01" ;
```

```
        :TimeOfGame "02:00:01" ;
        :StadiumCode 1 ;
        :SportCode 1 ;
        :Judge 1 .
<Game:2>
        :GameLevel "World" ;
        :DateOfGame "1970-01-01" ;
        :TimeOfGame "02:00:01" ;
        :StadiumCode 2 ;
        :SportCode 2 ;
        :Judge 2 .
<Participates:1>
        :GameCode 1 ;
        :AthleteCode 1 ;
        :Performance 9.76 ;
        :Valid "true" .
<Participates:2>
        :GameCode 1 ;
        :AthleteCode 3 ;
        :Performance 9.98 ;
        :Valid "true" .
<Participates:3>
        :GameCode 1 ;
        :AthleteCode 4 ;
        :Performance 9.45 ;
        :Valid "true" .
<Participates:4>
        :GameCode 2 ;
        :AthleteCode 4 ;
        :Performance 19.56 ;
        :Valid "true" .
<Stadium:1>
        :Name "Marakana" ;
        :Capacity "120000" ;
        :City "Rio De Janeiro" .
<Stadium:2>
        :Name "Anfield" ;
        :Capacity "75000" ;
        :City "Liverpool" .
<Judge:1>
        :Name "Petros" ;
        :Surnname "Petridis" ;
        :Genre "Male" ;
        :DateOfBirth "1970-01-01" .
<Judge:2>
        :Name "Ioanna" ;
        :Surnname "Ioannidou" ;
        :Genre "Female" ;
        :DateOfBirth "1970-01-01" .
```

```
<Volunteer:1>
        :Name "Giorgos" ;
        :Surnname "Georgiadis" ;
        :Genre "Male" ;
        :Age "23" ;
        :AT "A313136" .
<Volunteer:2>
        :Name "Maria" ;
        :Surnname "Papadopoulou" ;
        :Genre "Female" ;
        :Age "21" ;
        :AT "A213318" .
<Helps:1>
        :GameCode 1 ;
        :VolunteerCode 1 .
<Helps:2>
        :GameCode 2 ;
        :VolunteerCode 2 .
```

Σχήμα Σχεσιακής Βάσης Δεδομένων Ολυμπιακών Αγώνων

Εντολές Δημιουργίας Πινάκων	**Επεξήγηση**
CREATE TABLE VOLUNTEER (at char(10) not null primary key, name char(20), surname char(30), gender char(6), age integer);	Εθελοντής (Αστυνομική Ταυτότητα, Όνομα, Επώνυμο, Φύλο, Ηλικία)
CREATE TABLE STADIUM (name char(30) not null primary key, capacity integer, city char(20));	Στάδιο (Όνομα, Χωρητικότητα, Πόλη)
CREATE TABLE JUDGE (code integer not null primary key, name char(20), surname char(30), gender char(6), date_of_birth date);	Κριτής (Κωδικός, Όνομα, Επώνυμο, Φύλο, Ημερομηνία Γέννησης)
CREATE TABLE TRAINER (code integer not null primary key, name char(20), surname char(30), country_of_origin char(20));	Προπονητής (Κωδικός, Όνομα, Επώνυμο, Χώρα Καταγωγής)
CREATE TABLE SPORT (code integer not null primary key, name char(50),	Άθλημα (Κωδικός, Όνομα, Φύλο, Ρεκόρ, Ημερομηνία Ρεκόρ, Επώνυμο Ολυμπιονίκη, Όνομα Ολυμπιονίκη)

```
gender char(6),
record float,
record_date date,
recordman_surname char(30),
recordman_name char(20));

CREATE TABLE ATHLETE (
code integer not null primary key,
name char(50),
surname char(30),
gender char(6),
date_of_birth date,
weight float,
height integer,
country_of_origin char(20),
country_of_participation char(20),
trainer_code integer,
foreign key(trainer_code) references
TRAINER(code));

CREATE TABLE GAMES (
code integer not null primary key,
level char(20),
gdate date,
gtime time,
stadium char(30) not null references
STADIUM(name),
sport integer not null references SPORT(code),
judge integer not null references JUDGE(code));

CREATE TABLE HELPS (
game integer not null references GAMES(code),
volunteer char(10) not null references
VOLUNTEER(at),
primary key(game, volunteer));

CREATE TABLE PARTICIPATES (
game integer not null references GAMES(code),
athlete integer not null references
ATHLETE(code),
primary key(game, athlete),
performance float,
cancellation char(1));
```

Αθλητής (Κωδικός, Όνομα, Επώνυμο, Φύλο, Ημερομηνία Γέννησης, Βάρος, Ύψος, Χώρα Καταγωγής, Χώρα με την οποία συμμετέχει, Κωδικός Προπονητή)

Αγώνες (Κωδικός, Επίπεδο, Ημερομηνία Αγώνα, Ώρα Αγώνα, Όνομα Σταδίου, Κωδικός Αθλήματος, Κωδικός Κριτή)

Βοηθάει (Κωδικός Αγώνα, Ταυτότητα Εθελοντή)

Συμμετέχει (Κωδικός Αγώνα, Κωδικός Αθλητή, Καλύτερη Επίδοση, Έγκυρος ή Ακύρος)

RDF/XML *Syntax* Specification (Revised)

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .

<http://www.w3.org/1999/02/22-rdf-syntax-ns#> a owl:Ontology ;
        dc:title "The RDF Concepts Vocabulary (RDF)" ;
        dc:description "This is the RDF Schema for the RDF vocabulary
terms in the RDF Namespace, defined in RDF 1.1 Concepts." .
```

```
rdf:HTML a rdfs:Datatype ;
        rdfs:subClassOf rdfs:Literal ;
        rdfs:isDefinedBy <http://www.w3.org/1999/02/22-rdf-syntax-ns#> ;
        rdfs:seeAlso <http://www.w3.org/TR/rdf11-concepts/#section-html>
;
        rdfs:label "HTML" ;
        rdfs:comment "The datatype of RDF literals storing fragments of
HTML content" .

rdf:langString a rdfs:Datatype ;
        rdfs:subClassOf rdfs:Literal ;
        rdfs:isDefinedBy <http://www.w3.org/1999/02/22-rdf-syntax-ns#> ;
        rdfs:seeAlso <http://www.w3.org/TR/rdf11-concepts/#section-Graph-
Literal> ;
        rdfs:label "langString" ;
        rdfs:comment "The datatype of language-tagged string values" .

rdf:PlainLiteral a rdfs:Datatype ;
        rdfs:isDefinedBy <http://www.w3.org/1999/02/22-rdf-syntax-ns#> ;
        rdfs:subClassOf rdfs:Literal ;
        rdfs:seeAlso <http://www.w3.org/TR/rdf-plain-literal/> ;
        rdfs:label "PlainLiteral" ;
        rdfs:comment "The class of plain (i.e. untyped) literal values,
as used in RIF and OWL 2" .

rdf:type a rdf:Property ;
        rdfs:isDefinedBy <http://www.w3.org/1999/02/22-rdf-syntax-ns#> ;
        rdfs:label "type" ;
        rdfs:comment "The subject is an instance of a class." ;
        rdfs:range rdfs:Class ;
        rdfs:domain rdfs:Resource .

rdf:Property a rdfs:Class ;
        rdfs:isDefinedBy <http://www.w3.org/1999/02/22-rdf-syntax-ns#> ;
        rdfs:label "Property" ;
        rdfs:comment "The class of RDF properties." ;
        rdfs:subClassOf rdfs:Resource .

rdf:Statement a rdfs:Class ;
        rdfs:isDefinedBy <http://www.w3.org/1999/02/22-rdf-syntax-ns#> ;
        rdfs:label "Statement" ;
        rdfs:subClassOf rdfs:Resource ;
        rdfs:comment "The class of RDF statements." .

rdf:subject a rdf:Property ;
        rdfs:isDefinedBy <http://www.w3.org/1999/02/22-rdf-syntax-ns#> ;
        rdfs:label "subject" ;
        rdfs:comment "The subject of the subject RDF statement." ;
        rdfs:domain rdf:Statement ;
```

```
        rdfs:range rdfs:Resource .

rdf:predicate a rdf:Property ;
        rdfs:isDefinedBy <http://www.w3.org/1999/02/22-rdf-syntax-ns#> ;
        rdfs:label "predicate" ;
        rdfs:comment "The predicate of the subject RDF statement." ;
        rdfs:domain rdf:Statement ;
        rdfs:range rdfs:Resource .

rdf:object a rdf:Property ;
        rdfs:isDefinedBy <http://www.w3.org/1999/02/22-rdf-syntax-ns#> ;
        rdfs:label "object" ;
        rdfs:comment "The object of the subject RDF statement." ;
        rdfs:domain rdf:Statement ;
        rdfs:range rdfs:Resource .

rdf:Bag a rdfs:Class ;
        rdfs:isDefinedBy <http://www.w3.org/1999/02/22-rdf-syntax-ns#> ;
        rdfs:label "Bag" ;
        rdfs:comment "The class of unordered containers." ;
        rdfs:subClassOf rdfs:Container .

rdf:Seq a rdfs:Class ;
        rdfs:isDefinedBy <http://www.w3.org/1999/02/22-rdf-syntax-ns#> ;
        rdfs:label "Seq" ;
        rdfs:comment "The class of ordered containers." ;
        rdfs:subClassOf rdfs:Container .

rdf:Alt a rdfs:Class ;
        rdfs:isDefinedBy <http://www.w3.org/1999/02/22-rdf-syntax-ns#> ;
        rdfs:label "Alt" ;
        rdfs:comment "The class of containers of alternatives." ;
        rdfs:subClassOf rdfs:Container .

rdf:value a rdf:Property ;
        rdfs:isDefinedBy <http://www.w3.org/1999/02/22-rdf-syntax-ns#> ;
        rdfs:label "value" ;
        rdfs:comment "Idiomatic property used for structured values." ;
        rdfs:domain rdfs:Resource ;
        rdfs:range rdfs:Resource .

rdf:List a rdfs:Class ;
        rdfs:isDefinedBy <http://www.w3.org/1999/02/22-rdf-syntax-ns#> ;
        rdfs:label "List" ;
        rdfs:comment "The class of RDF Lists." ;
        rdfs:subClassOf rdfs:Resource .

rdf:nil a rdf:List ;
        rdfs:isDefinedBy <http://www.w3.org/1999/02/22-rdf-syntax-ns#> ;
```

```
        rdfs:label "nil" ;
        rdfs:comment "The empty list, with no items in it. If the rest of
a list is nil then the list has no more items in it." .

rdf:first a rdf:Property ;
        rdfs:isDefinedBy <http://www.w3.org/1999/02/22-rdf-syntax-ns#> ;
        rdfs:label "first" ;
        rdfs:comment "The first item in the subject RDF list." ;
        rdfs:domain rdf:List ;
        rdfs:range rdfs:Resource .

rdf:rest a rdf:Property ;
        rdfs:isDefinedBy <http://www.w3.org/1999/02/22-rdf-syntax-ns#> ;
        rdfs:label "rest" ;
        rdfs:comment "The rest of the subject RDF list after the first
item." ;
        rdfs:domain rdf:List ;
        rdfs:range rdf:List .

rdf:XMLLiteral a rdfs:Datatype ;
        rdfs:subClassOf rdfs:Literal ;
        rdfs:isDefinedBy <http://www.w3.org/1999/02/22-rdf-syntax-ns#> ;
        rdfs:label "XMLLiteral" ;
        rdfs:comment "The datatype of XML literal values." .
```