ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ, 6ο Εξάμηνο, ΣΗΜΜΥ ΕΜΠ 2018-2019 Εαρινό

Αναφορά εξαμηνιαίας εργασίας για την κατασκευή συστήματος βάσης δεδομένων για βιβλιοθήκη.

Ομάδα 68:

- Αθανάσιος Πέππας, ΑΜ = 03115749
- Κωνσταντίνος Κούστας, ΑΜ = 03115179
- Θρασύβουλος Ηλιάδης, ΑΜ = 03115761

Μέρος 1ο: Σχολιασμός διαδικασίας κατασκευής

- **Σύνοψη δομής project**

  Για το στήσιμο της βάσης και του UI χρησιμοποιήσαμε την προσέγγιση LAMP

  - Linux περιβάλλον
  - Apache Server
  - Mysql RDBMS
  - PHP

  Παρακάτω αναφέρονται πιο συγκεκριμένα οι τεχνολογίες που χρησιμοποιήσαμε όπως και οι σχεδιαστικές μας επιλογές.

- **Επιλογή περιβάλλοντος ανάπτυξης**

  Μετά από μια έρευνα στο διαδίκτυο μειώσαμε τις επιλογές μας στα πιο διάσημα περιβάλλοντα ανάπτυξης συστημάτων βάσεων δεδομένων του κόσμου. Ενδεικτικά μερικά ήταν:

  - MYSQL
  - Oracle NoSQL
  - PostgreSQL
  - SQL Server

  Γρήγορα απορρίψαμε την Oracle NoSQL για 1 κύριο λόγο, που είναι οι περιορισμένες δυνατότητες που θεωρήσαμε ότι προσφέρει η Basic έκδοση του λογισμικού. Ένας δευτερεύον λόγος είναι το μικρότερο μέγεθος του online community που ασχολείται με το NoSQL.

  Το project μας τελικά χρησιμοποιεί το περιβάλλον ανάπτυξης που προσφέρει η **Mysql**. Ο βασικός λόγος που την επιλέξαμε ήταν το εξαιρετικό documentation που υπάρχει online για το σύστημα. Η σουίτα θα λέγαμε mysql αποτελεί λογισμοκό ανοιχτού κώδικα (open-source) πράγμα που μας αρέσει να στηρίζουμε. Σημαντικότερος όμως λόγος από τους παραπάνω υπήρξε πως ξέραμε ότι στα πλαίσια του μαθήματος και του project θα βρισκόμασταν αντιμέτωποι με πολλές δυσκολίες, ερωτήσεις και απορίες, και έτσι θα χρειαζόμασταν ένα μεγάλο community από ανθρώπους όπως εμείς που θα συζητούσε για αυτά τα προβλήματα στα δημοφιλή online site όπως π.χ. το *ευλογημένο* www.stackoverflow.com. Ενώ υπάρχουν άνθρωποι που ασχολούνται με το PostgreSQL θεωρήσαμε ότι το πλήθος τους δεν

συγκρίνεται με αυτό της mysql. Πράγματι κατα τη διάρκεια της κατασκευής απευθυνθήκαμε επιτυχημένα πολλές φορές σε παλιά post του stackoverflow αλλά και ξεκινήσαμε τα δικά μας που ίσως βοηθήσουν και άλλους! Τέλος αξίζει να αναφερθεί ότι η mysql είναι ευρέως διαδεδομένη όχι μόνο στην εκπαίδευση αλλά και στη βιομηχανία, άλλο ένα πλεονέκτημα στη χρήση της.

Με την επιλογή της mysql δεν συναντήσαμε όμως μόνο πλεονεκτήματα. Η σουίτα αυτή ενώ είναι πολύ διαδεδομένη έχει κάποιες βασικές ελλείψεις στις δυνατότητές της. Πολλές από τις δυνατότητες της δεν υποστηρίζονται για παλιές εκδόσεις του λογισμικού, όπως π.χ. διαφορές στα Stored Procedures που τελικά καταλήξαμε να μη χρησιμοποιήσουμε για αυτό το λόγο.

Το πιο σημαντικό υστέρημα που συναντήσαμε είναι το ότι η mysql δεν υποστηρίζει τα CHECK constraints της SQL, πράγμα που μας έφερε σε δυσκολίες. Τελικά καταλήξαμε να χρησιμοποιούμε triggers για κάθε CHECK που θέλαμε να κάνουμε.

Στο community φαινεται να υπάρχουν επίσης ζητήματα απόδοσης και αδειοδότησης (όπως με κάθε λογισμικό βέβαια), για τα πλαίσια της εργασίας αυτής όμως ξέραμε ότι δεν μας αφορούν τόσο.

- **Οργάνωση αρχείων και συνεργασία**

  Για την καλύτερη ροή της εργασίας μας αποφασίσαμε να στήσουμε ένα repository στο github για να οργανώσουμε τα αρχεία μας. Καθώς είμαστε 3 άτομα είναι σημαντικό οι αλλαγές που έκανε ο καθένας μας να είναι άμεσα διαθέσιμες στους υπολοίπους. Το github repository της ομάδας μας είναι δημόσιο και βρίσκεται εδώ:

  https://github.com/Thapep/database

- **Άλλες τεχνολογίες**

  Για την ανάπτυξη της βάσης χρησιμοποιήσαμε το *MySQL Workbench*, στο οποίο φτιάξαμε και τεστάραμε τη βάση.

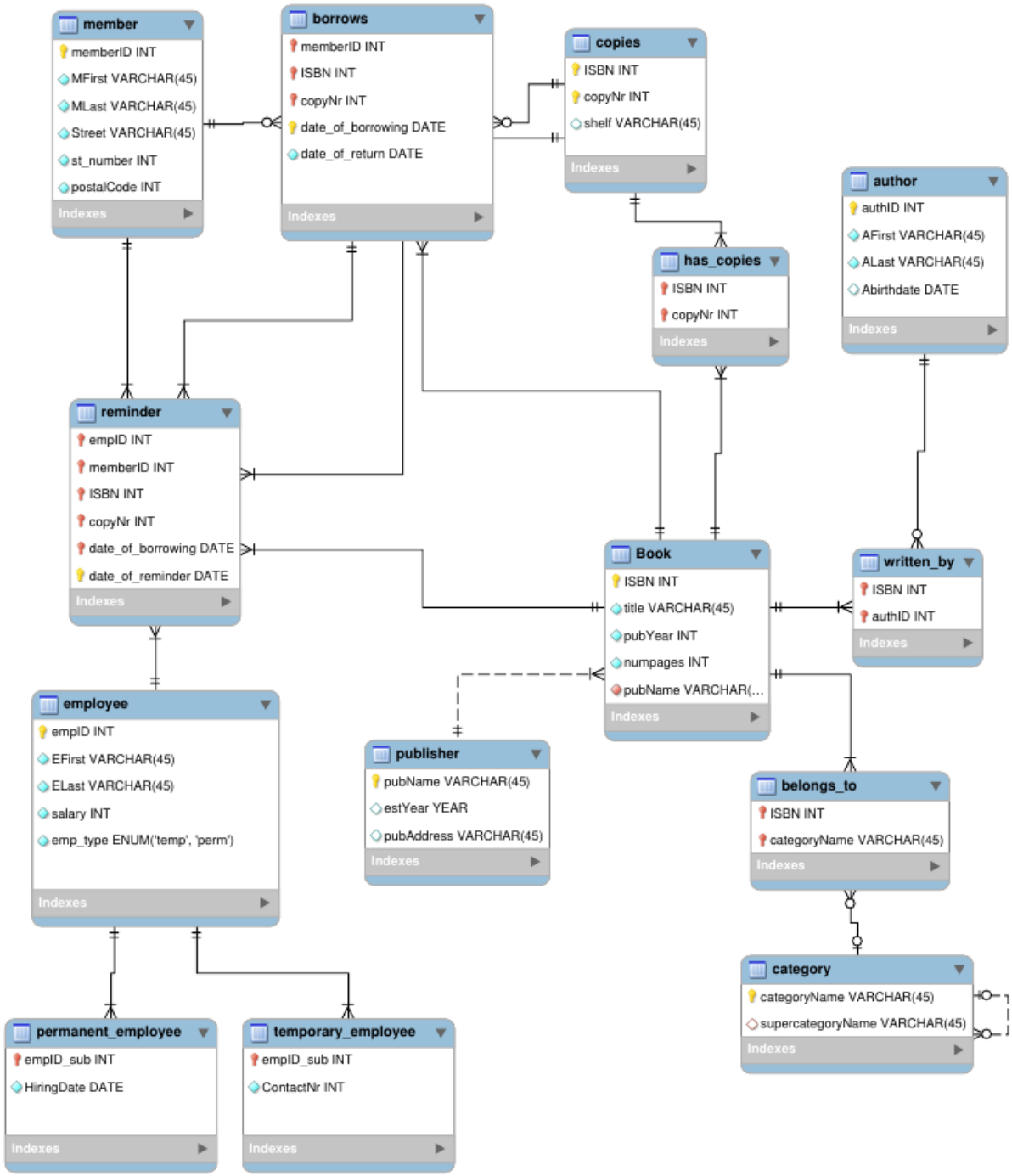
  Για την ανάπτυξη του User Interface χρησιμοποιήσαμε:

  - html/css/javascript για τη μορφοποίηση των σελίδων. Επίσης χρησιμοποιήσαμε στοιχεία από bootstrap για κάποιες σελίδες.
  - php για τη σύνδεση του UI με τη βάση.
  - sql σε κάποια αρχεία όπως το DDL, αλλά και σε άλλα που κρατήσαμε ως backup (π.χ. ο κώδικας για τα views και τα queries υπάρχει σε αποκλειστικά αρχεία .sql)

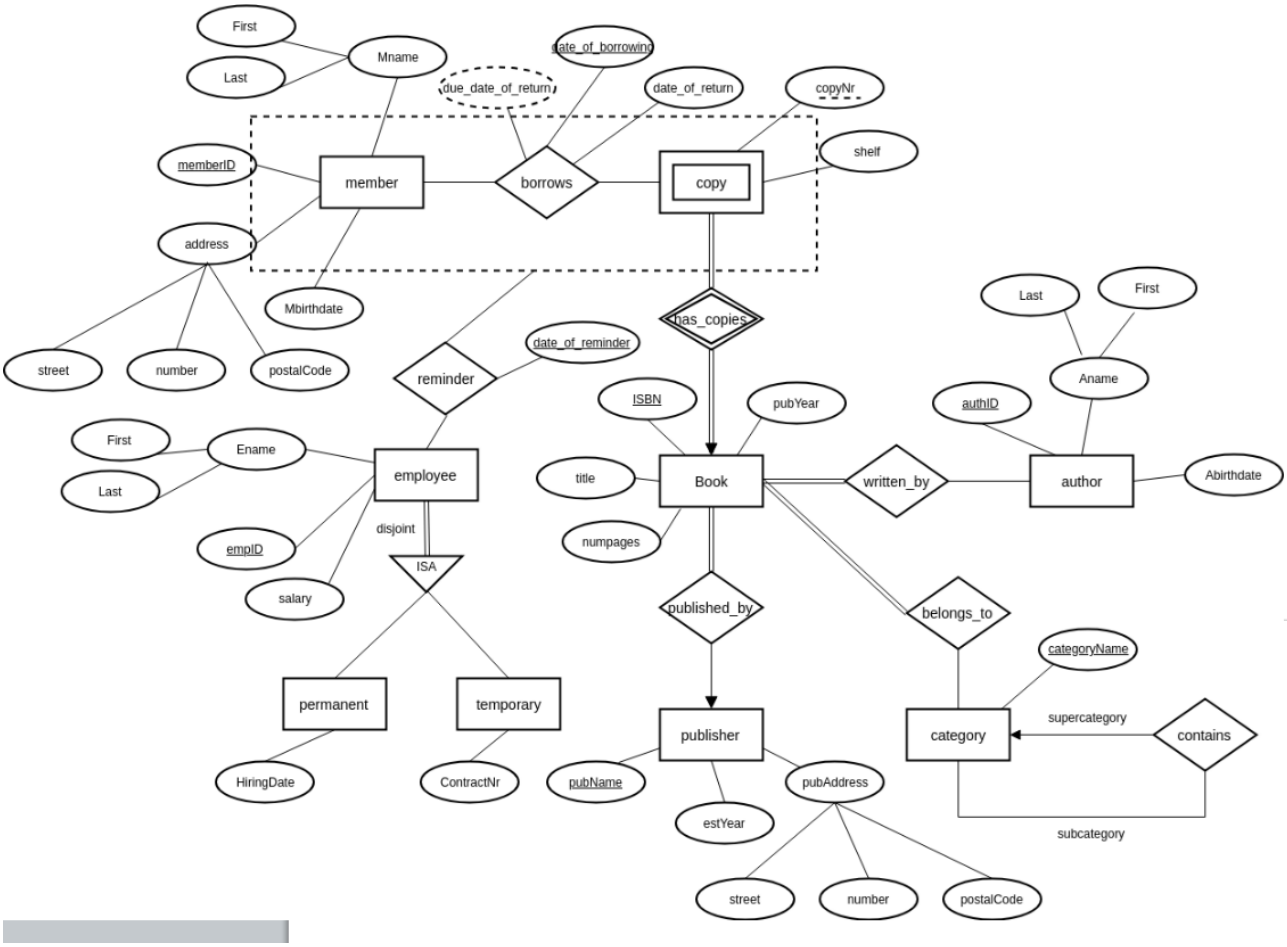
  Όλα τα παραπάνω αποτελούν τα standard της βιομηχανίας οπότε δε χρειάστηκε να κάνουμε κάποια σημαντική επιλογή.

Μέρος 2ο: Σχεδιασμός της Βάσης

Το διάγραμμα E-R της βάσης όπως αυτή προέκυψε στο MySQL Workbench είναι το εξής:



Η αντίστοιχη λύση που ακολουθήσαμε είναι η λύση της πρώτης άσκησης:



Για το σχεσιακό μοντέλο επιλέξαμε την προτεινόμενη λύση που μας δώσατε(η οποία ήταν αρκετά όμοια με τη δική μας λύση της 1ης άσκησης). Με βάση αυτή τη λύση σχεδιάσαμε το σχεσιακό μοντέλο στο *MySQL Workbench* και από αυτό πήραμε τη βάση.

Έτσι, χρησιμοποιήσαμε τους παρακάτω περιορισμούς ακεραιότητας(ICs):

- Καταρχάς, χρησιμοποιήσαμε τα ίδια *primary keys* καθώς και *foreign keys* όπως μας υποδείξατε. Συγκεκριμένα, έχουμε τα εξής primary keys:
    - memberID για το member
    - ISBN για το Book
    - authID για το author
    - categoryName για το category
    - (ISBN, copiesNr) για το copies
    - pubName για το publisher
    - empID για το employee
    - empID για το permanent_employee
    - empID για το temporary_employee
    - (memberID, ISBN, copyNr, date_of_borrowing) για το borrows
    - (ISBN, copyNr) για το has_copies
    - (ISBN, categoryName) για το belongs_to
    - (empID, memberID, ISBN, copyNr, date_of_borrowing, date_of_reminder) για το reminder
    - (ISBN, authID) για το written_by

    και τα εξής foreign keys:

    - pubName από το Book στο publisher

    - supercategoryName από το category στο category
    - empID από το permanent_employee στο employee
    - empID από το temporary_employee στο employee
    - memberID από το borrows στο member
    - ISBN από το borrows στο Book
    - (ISBN,copyNr) από το borrows στο copies
    - ISBN από το has_copies στο Book
    - (ISBN,copyNr) από το has_copies στο copies
    - ISBN από το belongs_to στο Book
    - categoryName από το belongs_to στο category
    - memberId από το reminder στο member
    - ISBN από το reminder στο Book
    - (ISBN, copyNr) από το reminder στο copies
    - empID από το reminder στο employee
    - (memberID, ISBN, copyNr, date_of_borrowing) από το reminder στο borrows
    - ISBN από το written_by στο Book
    - authID από το written_by στο author
- Για όλες τις σχέσεις που περιέχουν *foreign keys* ορίζουμε περιορισμούς ακεραιότητας ώστε να συμπληρώνονται αυτόματα τιμές στους δευτερεύοντες πίνακες κατά την εισαγωγή δεδομένων και να αποτρέπονται ενημερώσεις ή διαγραφές που είναι ανεπιθύμητες.

  Για τον λόγο αυτό, έχει προστεθεί ο περιορισμός *ON UPDATE CASCADE* ώστε, σε περίπτωση ανανέωσης μιας τιμής κάποιου πεδίου που αποτελεί πρωτεύον κλειδί, αυτή η αλλαγή να γίνει και στους υπόλοιπους πίνακες που έχουν το ίδιο πεδίο ως ξένο κλειδί. Επίσης, στις περισσότερες περιπτώσεις έχουμε προσθέσει τον περιορισμό ON *DELETE SET NULL* για να μη διαγράφονται δεδομένα που ενδεχομένως είναι απαραίτητα. Συγκεκριμένα, αν υπάρχει ένα βιβλίο το οποίο έχει δανειστεί και δεν έχει επιστρέψει κάποιο μέλος τότε δε γίνεται να διαγραφεί από τον πίνακα Book αφού υπάρχει ακόμα στον πίνακα borrows. Όμοια, δε γίνεται να διαγραφεί κάποιος συγγραφέας του οποίου βιβλίο υπάρχει ακόμα στο σύστημα(στον πίνακα written_by). Από την άλλη, σε κάποιες περιπτώσεις χρησιμοποιούμε τον περιορισμό *ON DELETE CASCADE* όπου επιτρέπουμε διαγραφές σε όλους τους πίνακες που περιέχουν το δεδομένο προς διαγραφή. Για παράδειγμα, αν διαγραφεί κάποιο βιβλίο από τον πίνακα Book, τότε αυτόματα θα διαγραφεί και από τους πίνακες written_by και has_copies.
- Συνολικά για κάθε πίνακα έχουμε τα εξής constraints και types:
    - **Member**

      `memberID` INT NOT NULL, `MFirst` VARCHAR(45) NOT NULL, `MLast` VARCHAR(45) NOT NULL, `Street` VARCHAR(45) NOT NULL, `st_number` INT NOT NULL, `postalCode` INT NOT NULL
    - **Book**

      `ISBN` INT NOT NULL, `title` VARCHAR(45) NOT NULL, `pubYear` INT NOT NULL, `numpages` INT NOT NULL, `pubName` VARCHAR(45) NOT NULL

      pubName(FK references publisher):

      ON DELETE NO ACTION

      ON UPDATE CASCADE
    - **author**

`authID` INT NOT NULL, `AFirst` VARCHAR(45) NOT NULL, `ALast` VARCHAR(45) NOT NULL, `Abirthdate` DATE NULL

- **category**

  `categoryName` VARCHAR(45) NOT NULL, `supercategoryName` VARCHAR(45) NULL DEFAULT 'N/A'

  supercategoryName(FK references category):

  ON DELETE NO ACTION

  ON UPDATE CASCADE

- **copies**

  `ISBN` INT NOT NULL, `copyNr` INT NOT NULL, `shelf` VARCHAR(45) NULL DEFAULT 'N/A'

- **publisher**

  `pubName` VARCHAR(45) NOT NULL, `estYear` YEAR NULL, `pubAddress` VARCHAR(45) NULL

- **employee**

  `empID` INT NOT NULL, `EFirst` VARCHAR(45) NOT NULL, `ELast` VARCHAR(45) NOT NULL, `salary` INT NOT NULL, `emp_type` ENUM('temp', 'perm') NOT NULL

- **permanent_employee**

  `empID_sub` INT NOT NULL, `HiringDate` DATE NOT NULL

  empID_sub(FK references employee):

  ON DELETE CASCADE

  ON UPDATE CASCADE

- **temporary_employee**

  `empID_sub` INT NOT NULL, `ContactNr` INT NOT NULL

  empID_sub(FK references employee):

  ON DELETE CASCADE

  ON UPDATE CASCADE

- **borrows**

  `memberID` INT NOT NULL, `ISBN` INT NOT NULL, `copyNr` INT NOT NULL, `date_of_borrowing` DATE NOT NULL, `date_of_return` DATE NOT NULL

  memberID(FK references member):

  ON DELETE NO ACTION

  ON UPDATE CASCADE

  ISBN(FK references Book):

  ON DELETE NO ACTION

  ON UPDATE CASCADE

  (ISBN,copyNr)(FK references copies):

  ON DELETE NO ACTION

  ON UPDATE CASCADE

- **has_copies**

`ISBN` INT NOT NULL, `copyNr` INT NOT NULL

ISBN(FK references Book):

ON DELETE CASCADE

ON UPDATE CASCADE

(ISBN,copyNr)(FK references copies):

ON DELETE CASCADE

ON UPDATE CASCADE

- **belongs_to**

  `ISBN` INT NOT NULL, `categoryName` VARCHAR(45) NOT NULL

  ISBN(FK references Book):

  ON DELETE CASCADE

  ON UPDATE CASCADE

  categoryName(FK references category):

  ON DELETE NO ACTION

  ON UPDATE CASCADE

- **reminder**

  `empID` INT NOT NULL, `memberID` INT NOT NULL, `ISBN` INT NOT NULL, `copyNr` INT NOT NULL, `date_of_borrowing` DATE NOT NULL, `date_of_reminder` DATE NOT NULL

  empID(FK references employee):

  ON DELETE CASCADE

  ON UPDATE CASCADE

  memberID(FK references member):

  ON DELETE NO ACTION

  ON UPDATE CASCADE

  ISBN(FK references Book):

  ON DELETE NO ACTION

  ON UPDATE CASCADE

  (ISBN,copyNr)(FK references copies):

  ON DELETE NO ACTION

  ON UPDATE CASCADE

  (memberID,ISBN,copyNr,date_of_borrowing)(FK references borrows):

  ON DELETE NO ACTION

  ON UPDATE CASCADE

- **written_by**

  `ISBN` INT NOT NULL, `authID` INT NOT NULL

  ISBN(FK references Book):

ON DELETE CASCADE

ON UPDATE CASCADE

authID(FK references author):

ON DELETE NO ACTION

ON UPDATE CASCADE

- Έχουν κατασκευαστεί αυτόματα κάποια *indexes* στη Βάση, κυρίως για τα *foreign keys* των πινάκων, τα οποία χρησιμοποιούνται συχνά για αναζήτηση, εισαγωγή, ενημέρωση ή διαγραφή.
- Επειδή η MySQL δεν υποστηρίζει ορισμένες λειτουργίες της SQL, όπως το check στα constraints και ο ορισμός disjoint για IS-A σχέσεις, ορισμένοι από τους περιορισμούς ακεραιότητας υλοποιούνται με triggers, όπως παρουσιάζονται στο 3ο μέρος.

Μέρος 3ο: DDL και queries βάσης

Παρακάτω παραθέτουμε τα DDL και όλα τα queries της βάσης. Σε σχόλια μέσα στον κώδικα θα βρείτε επεξηγήσεις για κάθε view, trigger και query που γράψαμε

Σημείωση: Τα tables δεν χρειάζονται περαιτέρω επεξήγηση καθώς το όνομά τους εξηγεί τη χρήση τους (και είναι και ίδια με τη λύση της άσκησης 1). Για περισσότερες όμως πληροφορίες δείτε το μέρος 2.

```
-- MySQL DDL for Library Database
-- This is the script we run to create the database to each new system. Because of
-- compatibility issues within the team, we sticked we the name 'mydb' for now

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8 ;
USE `mydb` ;

-- -----------------------------------------------------
-- Table `mydb`.`publisher`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`publisher` (
  `pubName` VARCHAR(45) NOT NULL,
  `estYear` YEAR NULL,
  `pubAddress` VARCHAR(45) NULL,
  PRIMARY KEY (`pubName`))
ENGINE = InnoDB;
```

```sql
-- -----------------------------------------------------
-- Table `mydb`.`Book`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Book` (
  `ISBN` INT NOT NULL,
  `title` VARCHAR(45) NOT NULL,
  `pubYear` INT NOT NULL,
  `numpages` INT NOT NULL,
  `pubName` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`ISBN`),
  INDEX `fk_Book_1_idx` (`pubName` ASC),
  CONSTRAINT `fk_Book_1`
    FOREIGN KEY (`pubName`)
    REFERENCES `mydb`.`publisher` (`pubName`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`member`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`member` (
  `memberID` INT NOT NULL,
  `MFirst` VARCHAR(45) NOT NULL,
  `MLast` VARCHAR(45) NOT NULL,
  `Street` VARCHAR(45) NOT NULL,
  `st_number` INT NOT NULL,
  `postalCode` INT NOT NULL,
  PRIMARY KEY (`memberID`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`author`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`author` (
  `authID` INT NOT NULL,
  `AFirst` VARCHAR(45) NOT NULL,
  `ALast` VARCHAR(45) NOT NULL,
  `Abirthdate` DATE NULL,
  PRIMARY KEY (`authID`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`category`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`category` (
  `categoryName` VARCHAR(45) NOT NULL,
  `supercategoryName` VARCHAR(45) NULL DEFAULT 'N/A',
  PRIMARY KEY (`categoryName`),
  INDEX `fk_category_category1_idx` (`supercategoryName` ASC),
```

```
  CONSTRAINT `fk_category_category1`
    FOREIGN KEY (`supercategoryName`)
    REFERENCES `mydb`.`category` (`categoryName`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`copies`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`copies` (
  `ISBN` INT NOT NULL,
  `copyNr` INT NOT NULL,
  `shelf` VARCHAR(45) NULL DEFAULT 'N/A',
  PRIMARY KEY (`ISBN`, `copyNr`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`employee`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`employee` (
  `empID` INT NOT NULL,
  `EFirst` VARCHAR(45) NOT NULL,
  `ELast` VARCHAR(45) NOT NULL,
  `salary` INT NOT NULL,
  `emp_type` ENUM('temp', 'perm') NOT NULL,
  PRIMARY KEY (`empID`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`written_by`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`written_by` (
  `ISBN` INT NOT NULL,
  `authID` INT NOT NULL,
  INDEX `fk_written_by_2_idx` (`authID` ASC),
  PRIMARY KEY (`ISBN`, `authID`),
  CONSTRAINT `fk_written_by_1`
    FOREIGN KEY (`ISBN`)
    REFERENCES `mydb`.`Book` (`ISBN`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_written_by_2`
    FOREIGN KEY (`authID`)
    REFERENCES `mydb`.`author` (`authID`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `mydb`.`belongs_to`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`belongs_to` (
  `ISBN` INT NOT NULL,
  `categoryName` VARCHAR(45) NOT NULL,
  INDEX `fk_belongs_to_2_idx` (`categoryName` ASC),
  PRIMARY KEY (`ISBN`, `categoryName`),
  CONSTRAINT `fk_belongs_to_1`
    FOREIGN KEY (`ISBN`)
    REFERENCES `mydb`.`Book` (`ISBN`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_belongs_to_2`
    FOREIGN KEY (`categoryName`)
    REFERENCES `mydb`.`category` (`categoryName`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`borrows`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`borrows` (
  `memberID` INT NOT NULL,
  `ISBN` INT NOT NULL,
  `copyNr` INT NOT NULL,
  `date_of_borrowing` DATE NOT NULL,
  `date_of_return` DATE NOT NULL,
  PRIMARY KEY (`copyNr`, `date_of_borrowing`, `ISBN`, `memberID`),
  INDEX `fk_member_has_copies_member1_idx` (`memberID` ASC),
  INDEX `fk_borrows_ISBN_idx` (`ISBN` ASC),
  INDEX `fk_borrows_ISBN_and_copyNr_idx` (`ISBN` ASC, `copyNr` ASC),
  INDEX `test` (`memberID` ASC, `ISBN` ASC, `copyNr` ASC, `date_of_borrowing` ASC),
  CONSTRAINT `fk_borrows_member1`
    FOREIGN KEY (`memberID`)
    REFERENCES `mydb`.`member` (`memberID`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_borrows_ISBN`
    FOREIGN KEY (`ISBN`)
    REFERENCES `mydb`.`Book` (`ISBN`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_borrows_ISBN_and_copyNr`
    FOREIGN KEY (`ISBN` , `copyNr`)
    REFERENCES `mydb`.`copies` (`ISBN` , `copyNr`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `mydb`.`permanent_employee`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`permanent_employee` (
  `empID_sub` INT NOT NULL,
  `HiringDate` DATE NOT NULL,
  PRIMARY KEY (`empID_sub`),
  CONSTRAINT `fk_permanent_employee_1`
    FOREIGN KEY (`empID_sub`)
    REFERENCES `mydb`.`employee` (`empID`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`temporary_employee`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`temporary_employee` (
  `empID_sub` INT NOT NULL,
  `ContactNr` INT NOT NULL,
  PRIMARY KEY (`empID_sub`),
  CONSTRAINT `fk_temporary_employee_1`
    FOREIGN KEY (`empID_sub`)
    REFERENCES `mydb`.`employee` (`empID`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`has_copies`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`has_copies` (
  `ISBN` INT NOT NULL,
  `copyNr` INT NOT NULL,
  PRIMARY KEY (`ISBN`, `copyNr`),
  CONSTRAINT `fk_has_copies_1`
    FOREIGN KEY (`ISBN`)
    REFERENCES `mydb`.`Book` (`ISBN`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_has_copies_2`
    FOREIGN KEY (`ISBN` , `copyNr`)
    REFERENCES `mydb`.`copies` (`ISBN` , `copyNr`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`reminder`
-- -----------------------------------------------------
```

```
CREATE TABLE IF NOT EXISTS `mydb`.`reminder` (
  `empID` INT NOT NULL,
  `memberID` INT NOT NULL,
  `ISBN` INT NOT NULL,
  `copyNr` INT NOT NULL,
  `date_of_borrowing` DATE NOT NULL,
  `date_of_reminder` DATE NOT NULL,
  PRIMARY KEY (`memberID`, `ISBN`, `copyNr`, `date_of_borrowing`, `empID`,
`date_of_reminder`),
  INDEX `fk_reminder_1_idx` (`empID` ASC),
  INDEX `fk_reminder_3_idx` (`ISBN` ASC),
  INDEX `fk_reminder_4_idx` (`ISBN` ASC, `copyNr` ASC),
  CONSTRAINT `fk_reminder_1`
    FOREIGN KEY (`empID`)
    REFERENCES `mydb`.`employee` (`empID`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_reminder_2`
    FOREIGN KEY (`memberID`)
    REFERENCES `mydb`.`member` (`memberID`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_reminder_3`
    FOREIGN KEY (`ISBN`)
    REFERENCES `mydb`.`Book` (`ISBN`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_reminder_4`
    FOREIGN KEY (`ISBN` , `copyNr`)
    REFERENCES `mydb`.`copies` (`ISBN` , `copyNr`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_reminder_5`
    FOREIGN KEY (`memberID` , `ISBN` , `copyNr` , `date_of_borrowing`)
    REFERENCES `mydb`.`borrows` (`memberID` , `ISBN` , `copyNr` , `date_of_borrowing`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;

USE `mydb` ;

-- -----------------------------------------------------
-- Placeholder table for view `mydb`.`oldest_books_10`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`oldest_books_10` (`ISBN` INT, `title` INT, `AFirst` INT,
`ALast` INT, `pubYear` INT);

-- -----------------------------------------------------
-- Placeholder table for view `mydb`.`member_names`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`member_names` (`memberID` INT, `MFirst` INT, `MLast`
INT);
```

```
-- -----------------------------------------------------
-- Placeholder table for view `mydb`.`author_list`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`author_list` (`'First Name'` INT, `'Last Name'` INT);

-- -----------------------------------------------------
-- Placeholder table for view `mydb`.`all_books`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`all_books` (`ISBN` INT, `'Title'` INT, `'Publication
Year'` INT, `'Publisher'` INT);

-- -----------------------------------------------------
-- View `mydb`.`oldest_books_10`
-- A view that returns the 10 oldest Books (according to the year of publication)
-- that the library posseses (either available or borrowed).
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`oldest_books_10`;
USE `mydb`;
CREATE  OR REPLACE VIEW `oldest_books_10` AS
    SELECT written_by.ISBN, Book.title, author.AFirst, author.ALast, Book.pubYear
    FROM written_by LEFT JOIN Book
    ON Book.ISBN=written_by.ISBN
    LEFT JOIN author ON author.authID=written_by.authID
    ORDER BY pubYear ASC
    LIMIT 10;

-- -----------------------------------------------------
-- View `mydb`.`member_names`
-- This view takes the member information that another member may like to know.
-- It returns only the member's Name and memberID, and it's purpose is to be
-- used by other members to inform them about the library community, without
-- disclosing personal information like the street they live in
-- or contact information.
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`member_names`;
USE `mydb`;
CREATE  OR REPLACE VIEW `member_names` AS
    SELECT memberID, MFirst, MLast
    FROM member
    ORDER BY memberID ASC;

-- -----------------------------------------------------
-- View `mydb`.`author_list`
-- Similarly to other views, the purpose of this one is to be used by Library users
-- to show them information about the authors whilst hiding information that are only
-- of use to the Library Administrators, such as author ID etc
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`author_list`;
USE `mydb`;
CREATE  OR REPLACE VIEW author_list AS
    SELECT author.AFirst as 'First Name', author.ALast as 'Last Name'
    FROM author;
```

```
-- -----------------------------------------------------
-- View `mydb`.`all_books`
-- A more convenient view for users to list all Books on the Library if needed.
-- This view is only created for the facilitation of the users and has no particular
-- role in any other operation.
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`all_books`;
USE `mydb`;
CREATE  OR REPLACE VIEW `all_books` AS
    SELECT
        ISBN,
        title AS 'Title',
        pubYear AS 'Publication Year',
        pubName AS 'Publisher'
    FROM
        Book;



SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Παραθέτουμε τώρα ένα αρχείο που περιέχει και επεξηγεί όλα τα triggers και τα queries που χρησιμοποιήθηκαν για τη βάση μας.

```
-- -----------------------------------------------------
--    TRIGGERS
-- -----------------------------------------------------


-- -----------------------------------------------------
-- Trigger `after_insert_employee`
-- This triggers assures of the IS-A disjoint relationship that the parent 'employee'
-- table has with the children 'temporary_employee' and 'permanent_employee'. Essentially
-- what this trigger does is it adds a new entry in the corresponding child table
-- according to the employee entry's field 'type' which is either 'perm' or 'temp'
-- thus ensuring that every entry in the employee table has a unique corresponding
-- entry in one of the child tables.
-- -----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE
DEFINER=`root`@`localhost`
TRIGGER `mydb`.`after_insert_employee`
AFTER INSERT ON `mydb`.`employee`
FOR EACH ROW
BEGIN
    IF NEW.emp_type = 'perm' THEN
```

```
        INSERT INTO permanent_employee (empID, HiringDate)
        VALUES(NEW.empID, null);
    ELSE
        INSERT INTO temporary_employee (empID, ContactNr)
        VALUES(NEW.empID, null);
    END IF;
END$$


DELIMITER ;

-- ----------------------------------------------------
-- Trigger `five_books_trigger`
-- This trigger acts as a 'CHECK' constraints because of the lack of mysql systems to
-- correctly parse the CHECK clause. Before any insert on borrows (i.e. before any member
-- borrows a book) this check runs, and only let's the insertion happen if the member
-- has not already borrowed (and not yet returned) 5 (or more) books. The implementation
-- we used was for the system to throw a user defined unhandled error, which is to be
-- handled by the equivalent php file.
-- ----------------------------------------------------

DELIMITER $$
CREATE TRIGGER five_books_trigger
BEFORE INSERT ON borrows
FOR EACH ROW
BEGIN
    IF  ((SELECT COUNT(*) FROM borrows WHERE member_memberID = NEW.member_memberID) >= 5)
THEN
        SIGNAL sqlstate '45000';
    END IF;
END $$

DELIMITER ;

-- ----------------------------------------------------
-- Creates a trigger that inserts a new copy in the 'copies' everytime a *new* Book is
-- added. The copyNr this trigger adds is always 1, as it is only called upon inserting
-- a new book (not a new copy) in the database.
-- ----------------------------------------------------

DELIMITER $$
CREATE TRIGGER after_insert_Book
AFTER INSERT ON Book
FOR EACH ROW
BEGIN
    INSERT INTO copies (ISBN, copyNr)
    VALUES(NEW.ISBN, 1);
END$$

DELIMITER;

-- ----------------------------------------------------
--    QUERIES
```

```sql
-- -------------------------------------------------------

-- -------------------------------------------------------
-- SQL QUERY `Books_and_authors_by_pubYear`
-- Returns all Library Books with their ISBN and Title, as well as author first and last
-- name. The query orders all books by publication year ascending. It is meant to be
-- used by library members or administrators for their own purposes. Available on the
-- UI via the Queries Menu page.
-- -------------------------------------------------------

SELECT written_by.ISBN, Book.title, author.AFirst, author.ALast
FROM written_by LEFT JOIN Book
ON Book.ISBN=written_by.ISBN
LEFT JOIN author ON author.authID=written_by.authID
ORDER BY pubYear ASC

-- -------------------------------------------------------
-- SQL QUERY `Books_by_publisher
-- Simple query that returns the amount of books from each publisher available in the
-- Library. Was implemented to showcase the use of aggregated functions in SELECT
-- statements as well as the use of the GROUP BY clause.
-- Available through the Queries Menu page.
-- -------------------------------------------------------

SELECT pubName as Publisher, COUNT(*) as 'Available Books' FROM Book
GROUP BY pubName;

-- -------------------------------------------------------
-- SQL QUERY `Books_by_supercategory`
-- Simiralry to the above SELECT statement, this returns the number of books in each
-- supercategory (such as 'Literature', 'Science' and so on). It was written to show
-- the use of nested JOIN statements in addition to the above.
-- Available through the Queries Menu page.
-- -------------------------------------------------------

SELECT supercategoryName as 'Supercategory', COUNT(book_belongs.title) as '# books in
supercategory' FROM (
    SELECT title, categoryName FROM Book JOIN belongs_to ON Book.ISBN = belongs_to.ISBN
    ) as book_belongs JOIN category
    ON book_belongs.categoryName = category.categoryName
GROUP BY supercategoryName
ORDER BY COUNT(book_belongs.title) DESC

-- -------------------------------------------------------
-- SQL QUERY `borrowed_books`
-- Returns all the currently borrowed books by ISBN and title as well as
-- the memberID and full name of the member in possesion of the book. Showcases
-- nested SELECT statements and is meant to be used by administrators in control
-- of the reminders.
-- Available through the Queries Menu page.
-- -------------------------------------------------------
```

```sql
SELECT borrows_member_copies.ISBN, Book.title, borrows_member_copies.memberID,
borrows_member_copies.MFirst, borrows_member_copies.MLast FROM (
    SELECT borrows_member.ISBN, borrows_member.memberID, borrows_member.MFirst,
borrows_member.MLast FROM (
        SELECT ISBN,member.memberID, member.MFirst, member.MLast, copyNr FROM borrows JOIN
member
        ON borrows.memberID = member.memberID
        ) as borrows_member
    JOIN copies ON borrows_member.ISBN = copies.ISBN and borrows_member.copyNr =
copies.copyNr
    ) as borrows_member_copies
JOIN Book ON borrows_member_copies.ISBN = Book.ISBN

-- -----------------------------------------------------
-- SQL QUERY `categories_per_supercategory_count`
-- Simple query to show number of categories inside a supercategory (e.g. 'Political
-- Science' and 'Mathematics' are subcategories to 'Science). Combined use of
-- aggregation functions, WHERE and GROUP BY clauses.
-- Available through the Queries Menu page.
-- -----------------------------------------------------

SELECT supercategoryName as 'Name', COUNT(supercategoryName) as 'Quantity'
FROM category
WHERE supercategoryName IS NOT NULL
GROUP BY supercategoryName

-- -----------------------------------------------------
-- SQL QUERY `number_of_books_after_year`
-- A query to show the 'history value' of the library. This returns the number books prior
-- or posterior to an input year. The below query is only an example of the use. The
-- correspondind php file let's the user give a year and choose prior/posterior to
-- fields to search for his own results.
-- Available through the Queries Menu page.
-- ** THIS IS ONLY AN EXAMPLE **
-- -----------------------------------------------------

SELECT pubYear, COUNT(pubName) FROM Book
GROUP BY pubYear
HAVING pubYear > 1900

-- -----------------------------------------------------
-- SQL QUERY `search_book`
-- A search book query that we implemented. This is an example that searches the title
-- of all books for any string *containing at any point* the string 'Manifesto'.
-- From the UI of course the user is allowed to give his own strings to search for.
-- Available from the index page as well as every other page on the UI.
-- ** THIS IS ONLY AN EXAMPLE **
-- -----------------------------------------------------

SELECT title, ISBN FROM Book
WHERE title LIKE "%Manifesto%";
```