**Imię i nazwisko: Aleksandra Jażdżewska**

**Indeks: 193601**

**Topic:**  1. Authors, books and publishers

**Task analysis:**

**Customer**: Highschool library in Chojnice

**User**: students and teachers

**Purpose**:

Keeping information about books in the library and their availability. Enabling student and teachers searching through library and borrowing books.

**Scenarios**

**Real life context:** Every student and teacher has their card registered in the system the moment they start studying or teaching at school. Everyone has only one card at time so when the card is lost then librarian blocks it in the system and create a new one for the user. When the person borrows the book (particular copy of a book), their card stores information about which book was borrowed and for how long (default time is 30 days). The deadline for giving the book back can be extended up to 14 days. With every day of overdue a fine is charged. Fine is charged also when the book is destroyed or lost. A user can search books based on genres, authors of book, publishers, title of book, title and authors of a particular writing or by filtering books based on the fact if they are set texts.

**Use cases:**

* Adding record of new user
* Adding card dedicated to specific user
* Searching through book that are set texts
* Searching through books that are of the same author or genre
* Blocking lost cards
* Inactivating cards of graduated students or fired/retired teachers
* Creating book event that describes lending out a book
* Reporting book being lost or destroyed
* Adding books, its copies, authors, genres and publishers
* Adding value of particular fines
* Using Fines table to balance sheet

**assumptions and limitations of the designed database:**

* All fines are schown in zloty currency.
* Only students and teachers can borrow books.
* If the list of set texts would be changed, then those changes would have to be made in the database.
* It could be extended by enabling users to log in. In that case type of users would have to be differentiated and new columns keeping logins and encrypted password would need to be done.
* Implementing many triggers increases computational complexity.
* Fines could be marked as paid using information from bank using task.

**inquiries to the database:**

* How many student from 3b have books overdue?
* How many copies of particular books are available to lend?
* How much is total fine of student with id 263?
* How many writings there are in a book with id 23?
* When did student with id 13 returned copy with id 387?
* How big is an overdue of book lend on card with id 57?
* How many books were destroyed in last year?

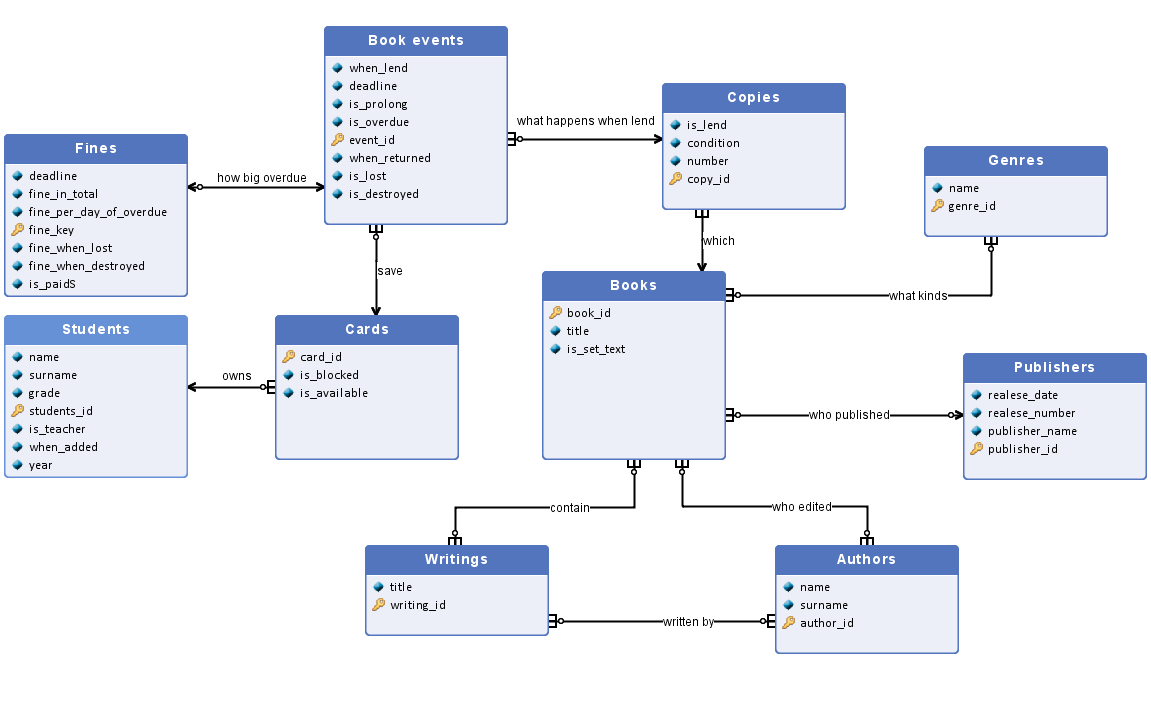
****

Figure 1ERD diagram of database for library

**Description of entities and attributes:**

**Entity set: Fines**Description: Records are created by a trigger which is executed when at least one of the attributes (is\_lost, is\_overdue, is\_destroyed) of book\_events is True. Then it stores fine which a particular user who borrowed particular book is charged with. The set is not deleted as it keeps the history of transactions that library made. ~hundreds

1. **deadline**: (date) NOT NULL   
   The date of return of book (YYYY, MM, DD). This attribute is needed here to calculate how big an overdue is. When attribute when\_returned in Book\_events is not null, then it stops calculating an overdue.
2. **fine\_in\_total**: (float (5,2)) default = 0

It sums all fines up. In case of overdue fine it multiplies value of fine\_per\_day\_of\_overdue times days after deadline. Days after deadline are calculated by taking date from when\_returned attribute in Books\_events and substracting deadline date from it. Cannot be negative.

1. **fine\_per\_day\_of\_overdue**: (float (5,2) ) default = 0.  
   It determines how big a fine of one day of overdue is. Cannot be negative or default = 0. Filled with the trigger (trigger fill it with the value of fine charged per day of overdue that is determined in library regulations 0.50).
2. **fine\_key**: (int (5)) PRIMARY KEY, UNIQUE, NOT NULL,   
   Auto Increment, starts with 0.
3. **fine\_when\_lost**: (float(5, 2)) default = 0  
   It determines how big a fine is when a book is lost. Cannot be negative. Filled with the trigger (trigger fill it with the value of fine charged for loosing book that is determined in library regulations – 30.00).
4. **fine\_when\_destroyed**: (float (5,2) ) default = 0  
   It determines how big a fine is when a book is destroyed. Cannot be negative. Filled with the trigger (trigger fill it with the value of fine charged for destroying a book that is determined in library regulations – 10.00).
5. **is\_paid**: (binary) default = False

Determines if fines are paid up or not. It enables user to pay after returning a book (so not in the same day).

**Entity set: Book events**Description: It describes what happens when a book (or a copy – to be exact) is lend so it registers information about the lending, such as date of lending and return or condition of book in the moment of returning. ~thousands

1. **when\_lend**: (current\_timestamp) NOT NULL

The date of lending out the book (YYYY MM DD).

1. **deadline**: (date) NOT NULL  
   The book must be returned until that date (YYYY MM DD). That date is automatically set to when\_lend date + 30 days.
2. **is\_prolong**: (binary) default = False  
   A student or teacher can prolong the book once and then the value of attribute changed from False to True. When is\_prolong is True, the trigger adds 14 days to the deadline.
3. **is\_overdue**: (binary) default = False  
   When the book is not returned until a deadline then it is overdue. Value is changed by librarian. When changed to True it triggers creating Fines record (triggers fill value of attribute fine\_per\_day\_of\_overdue).
4. **event\_id:**  (int(5)) PRIMARY KEY, UNIQUE, NOT NULL  
   Auto Increment, starts with 0.
5. **when\_returned**: (date) NOT NULL

The day when the book was actually returned (YYYY MM DD).

1. **is\_lost**: (binary) default = False

Becomes true when a student or teacher admits to losing the book. When changed to True creates record in Fines (triggers fill the value of Fine attribute: fine\_when\_lost).

**is\_destroyed**: (binary) default = False  
Becomes true when the book’s condition is worse than it was before lending. When changed to True creates record in Fines (triggers fill the value of Fine attribute: fine\_when\_destroyed).

**Entity set: Genres**Description: Table of books’ genres. It is created when a book fall into new genre. It might be deleted only when there is no book of a particular genre. ~15

1. **name**: ( varchar (30) ) NOT NULL  
   The type of genre for example fantasy or sci-fi. Only latin letters, no special characters.
2. **genre\_id**: (int (2) ) PRIMARY KEY

Auto Increment, starts with 0.

**Entity set: Students**Description: Table with the current and past users (students and teachers). New record is created when a new student or teacher is a school member. Record should be deleted to free up the space after 10 years of the date of when\_added when the past user is student or fired teacher. ~thousands

1. **name**: (varchar(30))

Name of the person. Only latin and polish letters.

1. **surname**: (varchar(30))

Surname of the person. Only latin and polish letters.

1. **grade**: ( enum (a,b,c,d, e,f) )

Determines in which class of particular year a student is. If is\_teacher is True, then the value is null.

1. **year**: (int(1)) NOT NULL  
   Determines in which year a student is (1st, 2nd, 3rd, 4th). Modified by librarian.
2. **students\_id**: (int(6)) PRIMARY KEY, UNIQUE, NOT NULL   
   Auto Increment, starts with 0.
3. **is\_teacher**: (binary)default = False  
   Checks if the person is a student or a teacher. If is\_teacher is True, then value of grade and year must be null.
4. **when\_added**: (date) NOT NULL  
   Date when a record of user was added. It is used to know in what years a past user was a member of school.

**Entity set: Cards**Description: Card enables user to lend a book as it has access to user’s data and then stores history of borrowing books by the owner. It is added when a new user is added or when a previous card of a user is lost. ~1200

1. **card\_id**: (int (6)) PRIMARY KEY, UNIQUE, NOT NULL   
   Auto Increment, starts with 0.
2. **is\_blocked**: (binary) default = False  
   If the owner of a card has lost it than their card can be blocked. The card can no longer be used (it no longer saves a history of lending a book).
3. **is\_available**: (binary) default = True  
   If the owner is fired or graduates the card is inactive.

**Entity set: Books**Description: Set of titles of books stored in library. The record is added when a library gets a new book/title. Record might be deleted only when there is no such title in library anymore. ~250

1. **book\_id**: (int (5)) PRIMARY KEY, UNIQUE, NOT NULL   
   Auto Increment, starts with 0.
2. **title**: (text(200))  
   A title of the book. Latin characters.
3. **is\_set\_text**: (binary) default = False   
   If True than book is an obligatory text, else it is not.

**Entity set: Publishers**Description: Set of publishers whose books library has. The record is added when there is a book of publisher that is not in the base yet. ~10

1. **release\_date**: (date) NOT NULL

Date when the book was released by the publisher (YYYY MM DD).

1. **release\_number**: (int(100))  
   Release number given by the publisher.
2. **publisher\_name**: (varchar(200))  
   A name of the publisher that released the book. Latin characters.
3. **publisher\_id**: (int (3)) PRIMARY KEY, UNIQUE, NOT NULL Auto Increment, starts with 0.

**Entity set: Authors**Description: Stores information about the author of a book or an author of a writing. The record is added when there is a book/writing written by an author that is not in the base yet. ~100

1. **name**: (varchar(100))

Name of the author. Latin characters.

1. **surname**: (varchar(200))  
   Surname of the author. Latin characters.
2. **author\_id**: (int(4)) PRIMARY KEY, UNIQUE, NOT NULL Auto increment, starts with 0.

**Entity set: Copies**Description: Set of copies of books stored in library. If a student wants to borrow a “Lalka” and library has 20 copies of “Lalka”, a student is going to borrow a copy, for example copy 1 and this entity describe this one particular copy. Record is added when a new book or copies or books are added to library. Record might be deleted only when it is no longer in library (or is no longer available to be lend) for example when a copy was lost or destroyed. ~5-30 copies of one book, ~800 in total

1. **is\_lend**: (binary) default = False  
   If False: particular copy of a book is currently available, else: particular copy of a book is currently lend.
2. **condition**: (varchar(500))  
   Stores information (simple comment) about the condition of a copy for example that it is brand new or that it misses one page, etc. Only latin characters.
3. **number**: (int(100)) NOT NULL  
   If a book has 20 copies, then this attribute says which one of this 20 is this copy, for example 2nd.
4. **copy\_id**: (int(5))PRIMARY KEY, UNIQUE, NOT NULL  
   Auto increment, starts with 0.

**Entity set: Writings**Description: Set of writings that can be found in books stored in library. Stores information about particular writing of an author for example “Tren V” by Jan Kochanowski. It can be found in many books. Student may want to read “Tren V” so they will need to borrow a book that contain this writing. It is added based on interests of students (books can contain many writings and it is hard to keep track of them all so the most popular and most often asked writings are added). Writing might be deleted only when all the books that contain it are deleted. ~900 in total

1. **title**: (char(200)  
   Title of a writing. Latin characters.
2. **writing\_id**: (int(4)) PRIMARY KEY, UNIQUE, NOT NULL   
   Auto increment, starts with 0.

**Description of Relationships:**

**save:**  Cards – Book\_events (1 : 0..n) Every lending of a book creates record of book\_events where action is connected with card of a lender. One card can save many events as one student can lend multiple books (but also can lend none) but a particular event can be saved only on one card.

**what\_kinds:** Genres – Books (0..n : 0..n) Book can have many genres as one book may be both sci-fi and romantic and also book can have none genres if the book is just added to database. One genre describe many books but also none when it was added in advance or when the only books of particular genre were deleted from the system.

**who\_published:** Books – Publishers (0..n : 0..1)  
 Books can have many publishers because book with title “Lalka” can be published by “Znak” and “czarne” publishers and can have none when it has just been added to the system. A publisher can publish many books and can have none assigned to when it just has been added to the database.

**who\_edited:** Authors – Books (0..n : 0..n)  
 Books can have one or many authors or none if author is not known or if the book has just been added. Author can write one or many books or none if they were added in advance or have just been added.

**how\_big\_overdue:** Book\_events – Fines (1 : 0..1) When attributes of Book\_events: is\_lost, is\_overdue, is\_destroyed are True then they trigger creating record of Fines. Book\_events may not have Fines when attributes above were False. Fines describes fines assigned to one particular lending of a book so a particular Fine can have only one particular Book\_events.

**owns**: Students – Cards (1 : 0..n) A person described by the Students needs to have a card described by the Cards to borrow a book. A card can have only one owner. A student/teacher can have one card at time, but if card is lost then they will have a new one, the previous one will be blocked (so all in all they can have multiple card over some period of time).

**what\_happens\_when\_lend:** Book\_events – Copies (0..n : 1)When new record of Book\_events is made then value of attribute of lend copy is changed to True. One copy can be lend multiple times so can have multiple book\_events but a particular event can store information about a particular lending of a copy.

**which:** Copies – Books (1..n : 1)One book have at least one copy, but a particular copy can be of only one particular book (20th copy of book “Harry Potter 1”).

**contain:** Writings – Books (0..n : 0..n)One book can contain many writings or none. One writing can be in many books. ( A writing “Sonet 120” by Petrarca can be found in both books “Best of Petrarca” and “Sonets” and book “Sonets” can have multiple writings such as “Sonet 120” and “Sonet 30”)

**written by:** Writings – Authors (0..n : 0..n)Writings can have none, one or multiple authors and authors can write multiple writings, but can also have none when they are just added to database.

Relational database schema

Fines (deadline, fine\_in\_total, fine\_per\_day\_of\_overdue, fine\_when\_lost, fine\_when\_destroyed, fine\_key, is\_paid, event\_id REF Book\_events)

Book\_events (when\_lend,  deadline, is\_prolong, is\_overdue, when\_returned, is\_lost, is\_destroyed, event\_id, card\_id REF Cards, copy\_id REF Copies)

Cards (in\_available, is\_blocked, card\_id, students\_id REF Students)

Students (name, surname, grade, year, is\_teacher, student\_id, When\_added) 

Copies (is\_lend, number, condition, copy\_id, book\_id REF Books)

Books (book\_id, title, is\_set\_text, publisher\_id REF Publishers)

Writings (title, writing\_id)

Authors (name, surname, author\_id)

Genres (name, genre\_id)

Publishers (release\_date, release\_number, publisher\_name, publisher\_id)

Contain (writing\_id REF Writings, book\_id REF Books)

Written\_by (writing\_id REF Writings, author\_id REF Authors)

Who\_edited (book\_id REF Books, author\_id REF Authors)

What\_kinds (book\_id REF Books, genre\_id REF Genres)