CS631: Fall 2017: Deliverable 3

Group 2: Zhenbo Qiao, Mike Lan, Leon Cheung

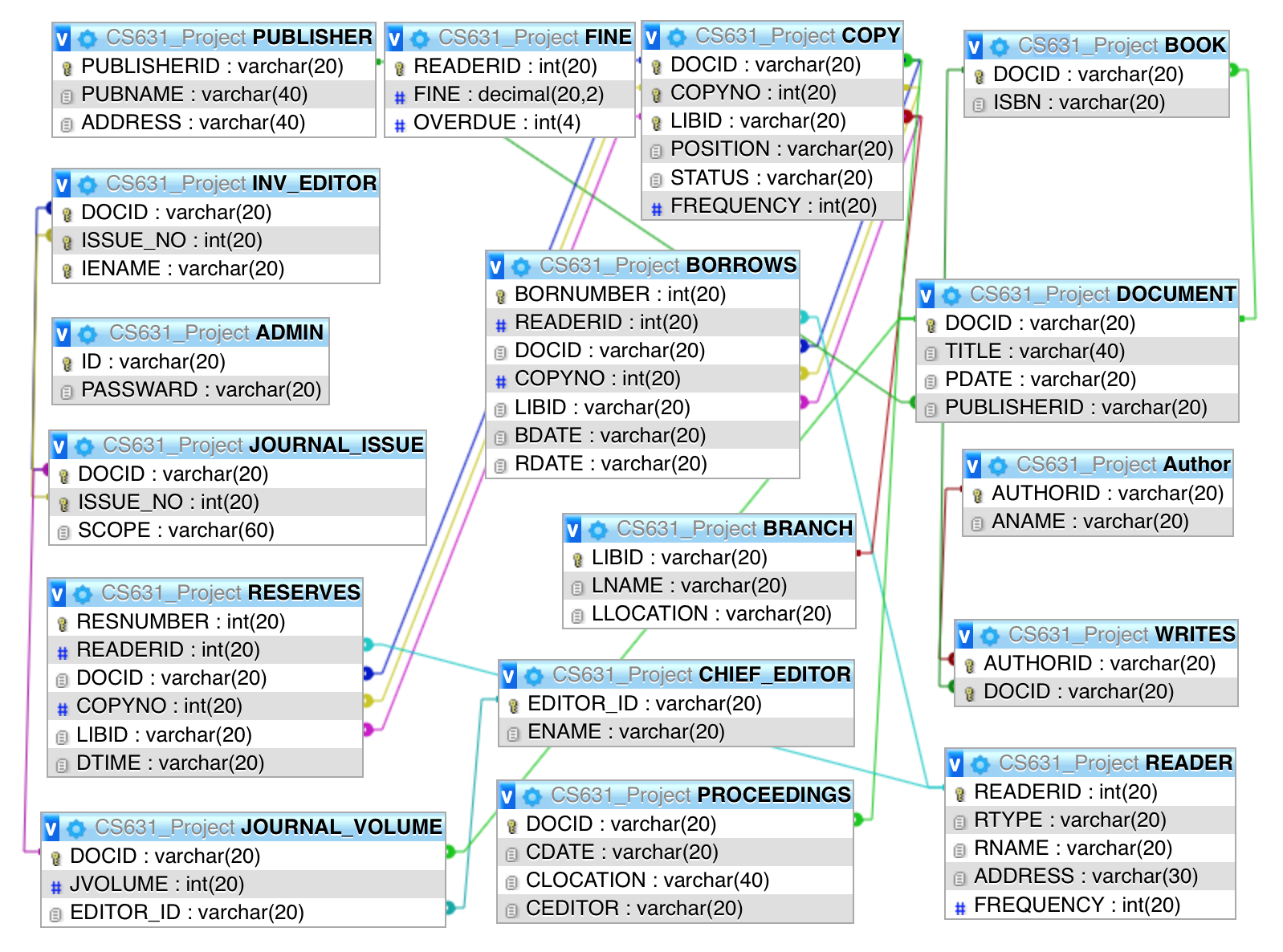
**Library Catalog: Documentation of implementation**

1. **GUI**

See user guide for in-depth description of GUI. The code was written in Python 2.7, and the GUI was constructed using the python libraries Tkinter, PIL and Datetime. The program has a controller class object that manages access to 3 class objects: StartPage, AdminMenu, and ReaderMenu.

**II. Documentation: Functions and SQL**

This application has many functions and methods with intuitive names that provide a glimpse of what kind of action it performs. These functions can be categorized into three components: Controller, Reader, and Administrator. The functions are as follows:

1. Controller
   1. dbConnect(sql\_input, query\_type): establishes a MySQL connection with user credentials and allow for SQL queries
   2. admin\_login()
      1. SELECT \* FROM ADMIN WHERE ID = '%s' AND PASSWARD = '%s' % (self.admin\_input.get(), self.passwd\_input.get())
   3. reader\_login()
      1. SELECT \* FROM READER WHERE READERID = '%s' % (self.cardnum\_input.get())
2. Reader/ Administrator
   1. print\_search()
      1. SELECT DOCID, COPYNO, LIBID, TITLE, PUBNAME, STATUS FROM DOCUMENT NATURAL JOIN COPY NATURAL JOIN PUBLISHER WHERE DOCID='%s' AND TITLE = '%s' AND PUBNAME ='%s' % (id\_query, title\_query, pub\_query)
      2. The print\_search function retrieves documents based on search the search value(s) entered which can be document ID, title, publisher name, or any combination of the three. It uses condition If statements to determine which search to perform. The application will display the results to the user with the following fields: ID, CopyNo, LibID, Title, Publisher, and Status
   2. update\_print\_reserved()
      1. After reader login, this method prints titles and status of currently reserved books for that particular user based on his/her readerid.
      2. SELECT TITLE, STATUS FROM DOCUMENT NATURAL JOIN RESERVES NATURAL JOIN COPY WHERE READERID ='%s' % (self.reader\_id)
   3. update\_print\_borrowed()
      1. After reader login, this method prints a list of titles, due dates, and fines (if any) for currently borrowed books by that particular reader.
      2. SELECT TITLE, BDATE, FINE FROM DOCUMENT NATURAL JOIN BORROWS WHERE READERID ='%s' AND RDATE<=>NULL" % (self.reader\_id)
   4. checkout()
      1. This method allows a logged in reader to checkout any available document copy or a copy reserved by that reader after inputting exact values for DOCID, COPYNO, and LIBID.
      2. This checks to see if the document copy exists in that library branch: SELECT \* FROM COPY WHERE DOCID = '%s' AND COPYNO = '%s' AND LIBID = '%s'" % (new\_id\_entry, new\_copy\_entry, new\_lib\_entry)
      3. This checks to see what the status of that particular document copy is: SELECT STATUS FROM COPY WHERE DOCID = '%s' AND COPYNO = '%s' AND LIBID = '%s'" % (new\_id\_entry, new\_copy\_entry, new\_lib\_entry)
      4. This checks whether document copy has been reserved by this reader: SELECT \* FROM RESERVES WHERE READERID='%s' AND DOCID='%s' AND COPYNO='%s' AND LIBID='%s'" % (self.reader\_id, new\_id\_entry, new\_copy\_entry, new\_lib\_entry)
      5. This adds the document copy to BORROWS table after verifying that it is available or reserved by that user for borrowing: INSERT INTO BORROWS (READERID, DOCID, COPYNO, LIBID, BDATE) VALUES ('%s', '%s', '%s', '%s', '%s')" % (self.reader\_id, new\_id\_entry, new\_copy\_entry, new\_lib\_entry, today)
      6. This SQL statement gets the frequency of which a particular document is borrowed so that it may be added to by the next SQL statement: SELECT FREQUENCY FROM COPY WHERE DOCID='%s' AND COPYNO='%s' AND LIBID='%s'" % (new\_id\_entry, new\_copy\_entry, new\_lib\_entry)
      7. This tallies the times a particular document copy from a certain branch is borrowed and sets the status to “borrowed” making it no longer available: UPDATE COPY SET STATUS='%s', FREQUENCY='%s' WHERE DOCID='%s' AND COPYNO='%s' AND LIBID='%s'" % ("Borrowed", str(freq\_borr+1), new\_id\_entry, new\_copy\_entry, new\_lib\_entry)
      8. Lastly, if the book was reserved and checked out, then it will be removed from reserves: DELETE FROM RESERVES WHERE READERID='%s' AND DOCID='%s' AND COPYNO='%s' AND LIBID='%s'" % (self.reader\_id, new\_id\_entry, new\_copy\_entry, new\_lib\_entry)
   5. reserve()
      1. This method allows a logged in reader to reserve any available document copy after inputting exact values for DOCID, COPYNO, and LIBID.
      2. This checks to see if the document copy exists in that library branch: SELECT \* FROM COPY WHERE DOCID = '%s' AND COPYNO = '%s' AND LIBID = '%s'" % (new\_id\_entry, new\_copy\_entry, new\_lib\_entry)
      3. After verifying it exists, this checks to see the status of the document : SELECT STATUS FROM COPY WHERE DOCID = '%s' AND COPYNO = '%s' AND LIBID = '%s'" % (new\_id\_entry, new\_copy\_entry, new\_lib\_entry)
      4. Only if it is available, then will it be reserved and inserted into table: INSERT INTO RESERVES (READERID, DOCID, COPYNO, LIBID, DTIME) VALUES ('%s', '%s', '%s', '%s', '%s')" %(self.reader\_id, new\_id\_entry, new\_copy\_entry, new\_lib\_entry, today)
      5. Lastly, it will need to be displayed that it is not available and is reserved: UPDATE COPY SET STATUS='%s' WHERE DOCID='%s' AND COPYNO='%s' AND LIBID='%s'" % ("Reserved", new\_id\_entry, new\_copy\_entry, new\_lib\_entry)
   6. return\_book()
      1. This function allows a logged in reader to return document(s) that have been borrowed by him/her after inputting exact values for DOCID, COPYNO, and LIBID.
      2. Checks to see if the document specified is indeed borrowed by user: SELECT \* FROM BORROWS WHERE READERID='%s' AND DOCID='%s' AND COPYNO='%s' AND LIBID='%s'" % (self.reader\_id, ret\_id\_entry, ret\_copy\_entry, ret\_lib\_entry)
      3. Checks to see the status of the document in COPY table; checking if it is borrowed: SELECT STATUS FROM COPY WHERE DOCID = '%s' AND COPYNO = '%s' AND LIBID = '%s'" % (ret\_id\_entry, ret\_copy\_entry, ret\_lib\_entry)
      4. After verification, then it will be updated in database, first in BORROWS: UPDATE BORROWS SET RDATE='%s' WHERE DOCID='%s' AND COPYNO='%s' AND LIBID='%s'" % (today, ret\_id\_entry, ret\_copy\_entry, ret\_lib\_entry)
      5. It then is updated in COPY table: UPDATE COPY SET STATUS='%s' WHERE DOCID='%s' AND COPYNO='%s' AND LIBID='%s'" % ("Available", ret\_id\_entry, ret\_copy\_entry, ret\_lib\_entry)
   7. logout()
      1. This lets a reader or administrator log out of application.
   8. view\_all()
      1. Displays all the documents and their information which includes ID, CopyNo, LibID, Title, Publisher, and Status.
      2. SELECT DOCID, COPYNO, LIBID, TITLE, PUBNAME, STATUS FROM DOCUMENT NATURAL JOIN COPY NATURAL JOIN PUBLISHER
   9. Add\_doc
      1. This method allows administrators to add document copies to the database
      2. Checks to see if entered document ID exists in the system: SELECT \* FROM DOCUMENT WHERE DOCID='%s'" % (new\_entry)
      3. Retrieves the last copy number for a document for a specific branch: SELECT MAX(COPYNO) FROM COPY WHERE DOCID='%s' AND LIBID='%s'" % (new\_entry, self.branch\_id.get())
      4. Lastly, this inserts the document copy with a 1 up number to copy number to that library branch: INSERT INTO COPY (DOCID, COPYNO, LIBID, POSITION, STATUS, FREQUENCY) VALUES ('%s', '%s', '%s', '%s', '%s', '%s')" %(new\_entry, str(resultCopyNo+1), self.branch\_id.get(), 1, "Available", 0)
   10. View\_by\_libid
       1. This method retrieves and displays document information by selected library branch and is used by administrators.
       2. SELECT DOCID, COPYNO, LIBID, TITLE, PUBNAME, STATUS FROM DOCUMENT NATURAL JOIN COPY NATURAL JOIN PUBLISHER WHERE LIBID ='%s'" % (self.branch\_id.get())
   11. Most\_freq\_borrowers
       1. This function allows the administrators to retrieve the names of the most frequent borrowers by branch in descending order.
       2. SELECT RNAME, COUNT(\*) FROM READER NATURAL JOIN BORROWS WHERE LIBID = '%s' GROUP BY LIBID, READERID ORDER BY READERID DESC" % self.branch\_id.get()
   12. Most\_borr\_books
       1. This function allows the administrators to retrieve the titles of the most borrowed books by branch in descending order.
       2. SELECT TITLE FROM BORROWS NATURAL JOIN DOCUMENT NATURAL JOIN BOOK WHERE LIBID = '%s' GROUP BY DOCID ORDER BY COUNT(\*) DESC" % self.branch\_id.get()
   13. Most\_pop\_books
       1. This function allows the administrators to retrieve the titles of the most popular or borrowed books by year in descending order regardless of library branch.
       2. SELECT TITLE FROM BORROWS NATURAL JOIN DOCUMENT NATURAL JOIN BOOK WHERE BDATE LIKE '2017%' GROUP BY DOCID ORDER BY COUNT(\*) DESC"
   14. Add\_reader
       1. This method allows administrators to add new readers to the database after filling in the corresponding fields of reader type, reader name, and address.
       2. INSERT INTO READER (RTYPE, RNAME, ADDRESS) VALUES ('%s', '%s', '%s')" %(new\_reader\_type\_input, new\_reader\_name\_input, new\_reader\_addr\_input)
   15. Update\_readers
       1. This method performs an average calculation of fines for each reader and displays the fines associated to a reader allowing the administrator to view average fines by reader.
       2. SELECT RNAME, FINE / OVERDUE AS AVERAGE FROM FINE NATURAL JOIN READER WHERE OVERDUE != 0 GROUP BY READERID"
3. **Database/data types/triggers/etc.**
4. Data type:
5. Triggers:
6. CREATE TRIGGER `ADDREADER` AFTER INSERT ON `READER`

FOR EACH ROW INSERT INTO `FINE` (`READERID`) VALUES (NEW.READERID)

1. CREATE TRIGGER `DELETEREADERID` AFTER DELETE ON `READER`

FOR EACH ROW DELETE FROM `FINE` WHERE READERID=OLD.READERID