WolfCity Publishing Management System

for WolfCity Publishing House

Project Report 3
CSC 540 Spring 2022

Andrew Abate, Chaitanya Patel, Morgan Chapman, Ilya Arakelyan
April 14, 2022

Assumptions

- ArticleText attribute of table Articles can be either a string containing article's text or a link to the actual text location in the file system (read-only) or can be empty.
- Distributors are paid (new invoice generated) on a monthly basis according to the production date ProduceByDate. The invoice amount (NumCopies*Price + Shipping) is aggregated over all orders fulfilled in a given month. The billing date is set as the first of the next month.
- For Distributor, a unique account number is assigned to each distributor, identifying them. Each distributor has a unique phone number, distributors can have the same name, type, and a street address, but not the same {street address, city} combination.
- For each order placed on behalf of a distributor, a unique order number, OrderID, is assigned. It determines the distributor name, a publication ordered, the number of copies, production date, price, and the shipping costs.
- If a distributor is removed from the Distributors table, its corresponding records are preserved in other tables to keep order history and financial information.
- For each Issue, there is a new Publication that is created, this is done because Issues cannot have a key by itself as it is only a subclass of Publication.
- ArticleText will be a link to the actual text location in the file system (read-only).
- If an issue does not have a specific title, an issue number or issue date should be used.
- Record deletion is allowed (for Publication/Issues/Books, Distributors, Employees) as per the project narrative. However, deletion of these items may affect some calculations in the monthly reports. Deletion of any Publication, Distributor, or Employee record that is referenced by the Payments, Orders, and Invoices relations is not recommended.
- Following the narrative, the Distribution Team can change all Distributor's attributes, including its outstanding balance. Being a more frequent and explicitly requested task, the Distribution Team changing only the Distributor's balance is included as an additional (though redundant) operation.
- A publication must be a periodical issue or a book edition.
- The Production section of the API handles the creation, updating, and deletion of all publications. Overlapping publication operations were excluded from Editing/Publishing.
- Periodicity is specified by the Publication type attribute. Books are published by edition; all journals are published monthly; all magazines are published weekly.
- A publication has an overall topic of interest. Journals and magazines also include articles with their own topic, which can be the same as the publication topic or different. A book only has a single topic.
- Authors of books are associated with the book edition as a whole, not just the chapter(s) for which the author is responsible.
- No chapters within a book edition may have the same title.
- No articles within a periodical issue may have the same title.
- Journalists are included in the authors table.
- As missing in the narrative, age and gender attributes in personnel are not included in the database (per Jiaqing Yuan - Wednesday, April 6, 2022 email).

1. Revisions

Report 1 Corrections

- On page 8, Section 5, Application Program Interfaces, subsection 4 Reports, totalPaymentsTime(employeeType, timePeriod) changed to totalPaymentsTime(employeeType, timeFrom, timeTo).

The change is included starting from Report 2.

Report 2 Corrections

- On page 10, we remove a constraint in initialization of Table Articles from ArticleText VARCHAR(128) NOT NULL to ArticleText VARCHAR(128) to match its description in Section 2 of page 6.

The change is included starting from Report 3.

2. Application Source Code

Application source code is submitted separately per instructions.

To run the application:

```
$ javac database_setup.java DBManager.java DBTablePrinter.java User.java
Reports.java Distribution.java Production.java Publishing.java Publisher.java
DistributionTeam.java Editor.java FinancialTeam.java

$ java database_setup
```

3. Transactions

Transaction 1: Enter a New Issue of a Publication

```
public void createIssue() {
        try {
                // start transaction
                db.disableAutocommit();
                System.out.println("CREATING ISSUE...");

                // get issue information from user
                System.out.print("Enter periodical title: ");
                String title = scanner.nextLine();
                System.out.print("Enter periodical type (Magazine or Journal): ");
                String type = scanner.nextLine();
                System.out.print("Enter issue topic: ");
                String topic = scanner.nextLine();
                System.out.print("Enter issue title: ");
                String issueTitle = scanner.nextLine();
                System.out.print("Enter issue date (YYYY-MM-DD): ");
```

```
                String issueDate = scanner.nextLine();
                System.out.print("Enter issue periodicity (Weekly/Monthly): ");
                String periodicity = scanner.nextLine();

                // INSERT INTO Publication
                String sql1 = String.format("insert into Publication values(NULL, '%s',
'%s', '%s');", title, type, topic);
                db.update(sql1);
                 // get auto-generated PublicationID
                result = db.query("select last_insert_id();");
                result.next();
                int pubID = result.getInt(1);

                // INSERT INTO Issues
                String sql2 = String.format("insert into Issues values(%d, '%s', '%s',
'%s');", pubID, issueTitle, issueDate, periodicity);
                db.update(sql2);

                // commit and display confirmation
                if (db.commit()) {
                        System.out.println("Successfully Added Following Record:");
                        String sql3 = String.format("select * from Publication natural join
Issues where PublicationID = %d;", pubID);
                        result = db.query(sql3);
                        DBTablePrinter.printResultSet(result);
                }

        } catch (Exception e) {
                e.printStackTrace();

        } finally {
                // enable auto-commit
                db.enableAutocommit();
        }
    }
```

Transaction 2: Create a New Employee - Editor

```
public void addEditor() {
        try {
                // start transaction
                db.disableAutocommit();

                // enter new values
                System.out.print("Enter name: ");
                String name = scanner.nextLine();
                System.out.print("Enter type (Staff/Invited): ");
                String type = scanner.nextLine();
                System.out.print("Enter Phone Number: ");
                String phone = scanner.nextLine();
                System.out.print("Enter Email: ");
```

```
                String email = scanner.nextLine();
                System.out.print("Enter Address: ");
                String address = scanner.nextLine();

                // insert into Employees table
                String sql = String.format("INSERT INTO Employees(Name, Type, Phone, Email,
Address)"
                                + "VALUES ('%s', '%s', '%s', '%s', '%s');", name, type, phone,
email, address);
                db.update(sql);

                // get ID of newly created employee
                result = db.query("SELECT last_insert_id();");
                result.next();
                int employeeID = result.getInt(1);

                // insert into Editors table
                sql = String.format("INSERT INTO Editors VALUES (%d);", employeeID);
                db.update(sql);

                // commit and print results
                result = db.query(String.format("SELECT * FROM Employees WHERE EmpID = %d;",
employeeID));
                if (db.commit()) {
                        System.out.println("\nSuccessfully Added Following Record");
                        DBTablePrinter.printResultSet(result);
                        System.out.println();
                }
                // end transaction
                db.enableAutocommit();
        } catch (Exception e) {
                e.printStackTrace();
        }
    }
```

Note: Transaction rollback functions are performed by DBManager (db object, here) if a commit was unsuccessful.

4. Documentation

Documentation is submitted as part of the source code in the JavaDoc format. An html index can be built by running:

```
$ javadoc database_setup.java DBManager.java DBTablePrinter.java User.java
Reports.java Distribution.java Production.java Publishing.java Publisher.java
DistributionTeam.java Editor.java FinancialTeam.java
```

Design Decisions

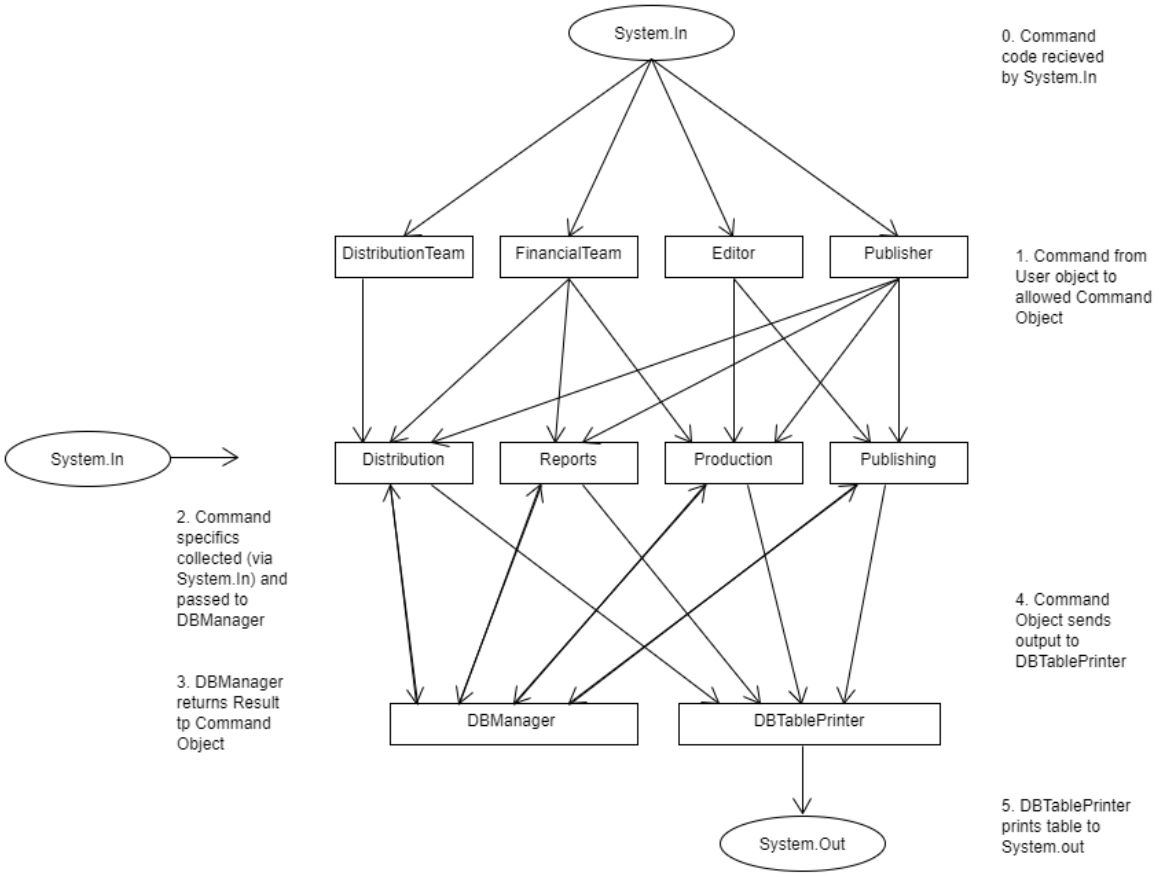
First part of the design was to include 2 loops: one for user selection/login and one for passing input commands to the selected user type. Command input from the application user starts in the database_setup file, the main driver for the program. There are 4 User types (DistributionTeam, FinancialTeam, Editor, and Publisher). There are 4 Command Objects that correspond to operations grouped together for similarity (Distribution, Reports, Production, and Publishing), as described in the project narrative. The DBManager object is the primary way for interacting with the Database. The DBTablePrinter is an open source file that prints out the result of a query.

After User type selection, the User object is passed the DBManager object and scanner object created upon program startup. The User object contains different command options that are available for each type of User to use. The User object then passes the selected command to the appropriate Command object. The command object uses the scanner object to receive and parse any specifics related to that command. Once all specifics are collected, an SQL statement is created and sent to the DBManager object. If auto-commit is disabled for the API operation, the DBManager handles each command as an individual transaction and commits at the end of the command. If the commit is unsuccessful, the DBmanager will perform a rollback of all statements within the transaction. If the transaction commits successfully, the result set is returned to the Command object. The Command object then sends the result set to the DBTablePrinter static object call to print the result out.

Notable design choices:

- The command ‘exit’ or ‘quit’ will close the DBManager connections and exit the program.
- The command ‘logout’ will return to the User login menu
- Any other inputs other than the command will print out a table of all available commands that User is allowed to use.
- DBTablePrinter is an open source file found at https://github.com/htorun/dbtableprinter

Command Flow Chart



Team Roles

Part 1:
Software Engineer: Ilya Arakelyan (Prime), Chaitanya Patel (Backup)
Database Designer/Administrator: Morgan Chapman (Prime), Andrew Abate (Backup)
Application Programmer: Andrew Abate (Prime), Morgan Chapman(Backup)
Test Plan Engineer:Chaitanya Patel (Prime), Ilya Arakelyan (Backup)

Part 2:
Software Engineer: Chaitanya Patel (Prime), Andrew Abate (Backup)
Database Designer/Administrator: Andrew Abate (Prime), Ilya Arakelyan (Backup)
Application Programmer: Ilya Arakelyan (Prime), Morgan Chapman (Backup)
Test Plan Engineer: Morgan Chapman (Prime), Chaitanya Patel (Backup)

Part 3:
Software Engineer: Morgan Chapman(Prime), Ilya Arakelyan (Backup)
Database Designer/Administrator:Chaitanya Patel (Prime), Andrew Abate (Backup)
Application Programmer: Ilya Arakelyan (Prime),Chaitanya Patel (Backup)
Test Plan Engineer: Andrew Abate(Prime), Morgan Chapman (Backup)