WolfPubDb Management System

For WolfCity Publishing House

CSC540 Database Management Systems

Project Report 1

Amol Gautam, Nodirbek Korchiev, Rahul Yedida, Vodelina Samatova

Date: 02-17-2020

Assumptions

1. Articles and books are stored online, and the database will only contain links to these documents for efficient storage, rather than storing the entire text.
2. Topics that publications are about have unique names, i.e., the name itself forms a primary key for the Topic entity set.
3. Each publication has a *cost price* and a *selling price*. The former is the price that the distributor buys at from the publishing house. The latter is the price the distributor sells at to the general public. This allows for the database to record the profits from each publication.
4. Distributors are uniquely identified by name.
5. Authors write individual articles or books, or some of both. However, an editor is responsible for collating various articles together into a bigger periodic publication (we assume “articles” do not exist independently), and “authors” as defined in our database do not write publications themselves.
6. Publications can have different dates of publication than the articles it contains. For example, research papers may be written at an earlier point in time, and perhaps released as a preprint. The journal issue still contains this same paper, but the issue itself is published at a later date.
7. Staff may be re-hired; for that reason, we include their start dates as part of the primary key.
8. Staff may be paid periodically or one-time. In the latter case, we simply set the periodicity attribute to 0.
9. The term “location” means street address.

1. Problem Description

WolfCity Publishing House is in need of a system to manage records of their system. At a high level, the system must be able to maintain information about publications, articles, and books. However, it must also contain information about the more intricate aspects of the publishing system: specifically, it must contain details of authors, editors, topics that publications cover, and details of the distributors that distribute publications, along with orders that they must be able to track. Presumably, distributors place these orders with the publishing house and may make partial payments, maintaining a balance.

We propose the use of a SQL database management system for this task. Through the constraints of the relational data model and integrity and uniqueness constraints enforceable by SQL, it is easy to build a maintainable, robust, consistent source of data for the entire publishing system. Multiple levels of sanity checks can be performed: at the client (user) level, at the server level, and at the database level. In a concrete implementation, these would be, respectively, a front-end GUI for the user, a server that accepts queries (possibly also providing a REST interface), and the DBMS itself.

2. System Users

Managers of the publishing house (and billing staff)

The managers (and billing staff) of the publishing house can see all the operations of the house, including distribution, authors, editors, etc. They have the most knowledge of the extent of the operations of the house.

Distributors

Distributors place orders of publications with the publishing house. Presumably, they can order by authors, or titles, and therefore can see them in their view.

Editors

Editors are responsible for collating various articles into publications. They also edit books. They know a fairly detailed view of the system, including issues, authors, and topics. They are, however, blind to the distribution system.

Authors

Authors write individual articles and books. They correspond with editors and can see the topics associated with their articles and books.

3. Five Main Entities

Book information: Publication ID (pid), Title, Publication Type (type), Date of Publication(dop), URL, Price, ISBN

Periodic Publication Information: Publication ID (pid), Title, Publication Type (type), Date of Publication (dop), URL, Price, periodicity, DOI, info

Staff information: Staff ID (SID), Staff name (name), Staff type(type), Start date (sdate), Date of Birth (dob), Phone number (phone), Gender

Distributor Information: Distributor ID (did), Distributor name (dname), Distributor type (dtype), City, Address (address), Contact information (contact), photo (phno)

Order information: Order ID (oid), Price, Number of copies (copies), Order date (odate)

4. Tasks and Operations- Realistic Situations

SITUATION 1

A distributor named Bob arrives at the publishing house and asks for the receptionist about the total balance and information on the last order. He notices that the publication house does not have his most recent address. He tells the receptionist to update the address so that it reflects his most recent address.

SITUATION 2

A writer named Alice arrives the publishing house and informs them about a new article that she has written and request the publishing house to publish it. The manager adds the article in the Publication relation and asks the editor to review the article.

5. API Design

The API is designed in the following way. IDs are auto-generated for entities; therefore, there are methods to get the ID given other attributes. The exception to this is for orders—the assumption here is that distributors remember the order numbers, for simplicity. There are individual methods to update attributes of entities. A question mark in the function signature means a nullable attribute. The assumption here is that these are passed to the server (which implements the following methods) in JSON format, so there is no necessity for all attributes to be present in the object. An example is for `createPublication`, where the server infers whether the publication is a question from the `type` attribute and decides whether or not to use the (potentially unavailable) `ISBN` attribute.

Note that the functions here are NOT the REST API endpoints; these are the APIs that are implemented in the server as an interface to the database system. We do not present a REST API here, but it should be trivial to generate one given the following API functions.

For security reasons, when staff members want to access details of publications, they must first sign in. When they do so, the server generates a token in the JSON Web Token (JWT) format (for details, see https://jwt.io). In brief, this token is simply a hashed JSON object that contains the username. Because of the way it is encoded (*not* encrypted), only the server can decode it. Therefore, when the user wishes to access the details of a publication, the client also sends in the request, the JWT (which is stored in the client's local storage). The server decodes the token, and if valid, checks whether the user is authorized to view the publication details. If so, it returns the details; otherwise, it returns a 403 Forbidden (in case of a web app; for other applications, it may send an appropriate "forbidden" response). This is why `getPublicationDetails` only includes the token (which stores the username) and the publication id—no other information is required.

Most of the functions described below return a Boolean. Where unspecified, it should be assumed that this Boolean represents the success of the operation, i.e., a true return value means a successful operation, and a false return value means a failure—in such cases, an explanation will also be returned by the server to the client in JSON.

Publications: also responsible for books

```
createPublication(title, type, pubDate, url, price, ISBN?,
periodicity?)→ bool
```

Auto-generates id and adds a new publication.

```
updatePublicationUrl(id, newUrl)→ bool
```

```
updatePublicationPrice(id, newPrice) → bool
```

```
findPublicationByDate(date) → string
```

Returns the url of the publication.

```
findPublicationByAuthor(author) → string
```

Returns the url of the publication.

```
getPublicationId(title, url) → int
```

Returns the publication id.

```
getPublicationDetails(token, pubId) → object
```

Assumes the use of JWT/other token; returns details if authorized.

```
getTotalCopiesSold(token, distId, pubId) → int

getTotalSales(token, distId, pubId) → double

getTotalRevenue(token) → double

getTotalExpenses(token) → double

deletePublicationById(pid) → bool
```

Distributors

```
createDistributor(name, type, city, phone, addr, contact)→ bool
```

Auto-generates id and sets balance to 0.

```
updateDistributorCity(id, city)→ bool

updateDistributorPhone(id, phone)→ bool

updateDistributorAddr(id, addr)→ bool

updateDistributorContact(id, contact) → bool

getDistributorId(name, phone) → int

deleteDistributor(id) → bool

getDistributorCount() → int
```

This is assumed to not be a protected item, hence no token is required.

`getDistributorRevenue(token, name) → double`

`getRevenueByCity(token, city, name) → double`

`getRevenueByAddr(token, addr, name) → double`

Orders

`createOrder(copies, price, shcost, date)→ bool`

Auto-generates id **and returns it**: there is no way to get it again.

`madeOrder(dId, oId) → bool`

Inserts an entry into the `MadeOrder` relation.

`createOrderPart(oId, pId)→ bool`

Inserts an entry into the `Contains` relation between `Publications` and `Order`.

`addBalance(dId, addition) → bool`

It is assumed that the addition can be negative as well, to reduce the balance.

Editors

`addStaff(name, phone, dob, type, startDate, pay, payPeriodicity)→ bool`

Auto-generates id

`fireStaffById(sid) → bool`

`getStaffId(name, phone, dob)→ int`

`setEditor(staffId, pubId) → bool`

`getStaffRevenue(token, staffId) → double`

`getAllStaffRevenueBetween(token, startDate, endDate) → double`

Topics

`addTopic(name)→ bool`

Auto-generates topic ID

`getTopicId(name) → int`

`setTopic(pId, tId) → bool`

Adds a topic with id `tId` to a publication with id `pId`.

`findByTopic(topic) → string`

Returns the url of the article.

Chapters

`createChapter(bookId, number, url, title)→bool`

Auto-generates id

`removeChapter(bookId, number)→ bool`

`setAuthor(pubId, type, staffId) → bool`

We also pass in where to look (type). This lets one method handle authors writing articles as well as books.

Articles

`createArticle(title, date, url) → bool`

Auto-generates id

`getArticleId(title, url) → int`

Auto-generates id

`setArticleParent(aId, pubId, issueNo) → bool`

Sets article with id `aId` to be a part of periodic publication issue (number `issueNo`) with id `pubId`.

`removeArticle(aId, pubId)→ bool`

Issue

`createIssue(number, pubId) → bool`

6. User Views

Managers of the publishing house

Managers can see the publications being printed, the articles collated by those publications, and the authors. They can set editors for publications, and view and modify the topics for each publication. Managers can see which distributor placed orders for which publications. Therefore, they have the widest view of the system.

Distributors

Distributors have a limited view of the system. They can see the orders they have placed with the publishing house, and place additional orders. They may make partial payments on orders and track their total balance. Finally, they may update the selling price of each publication.

Editors

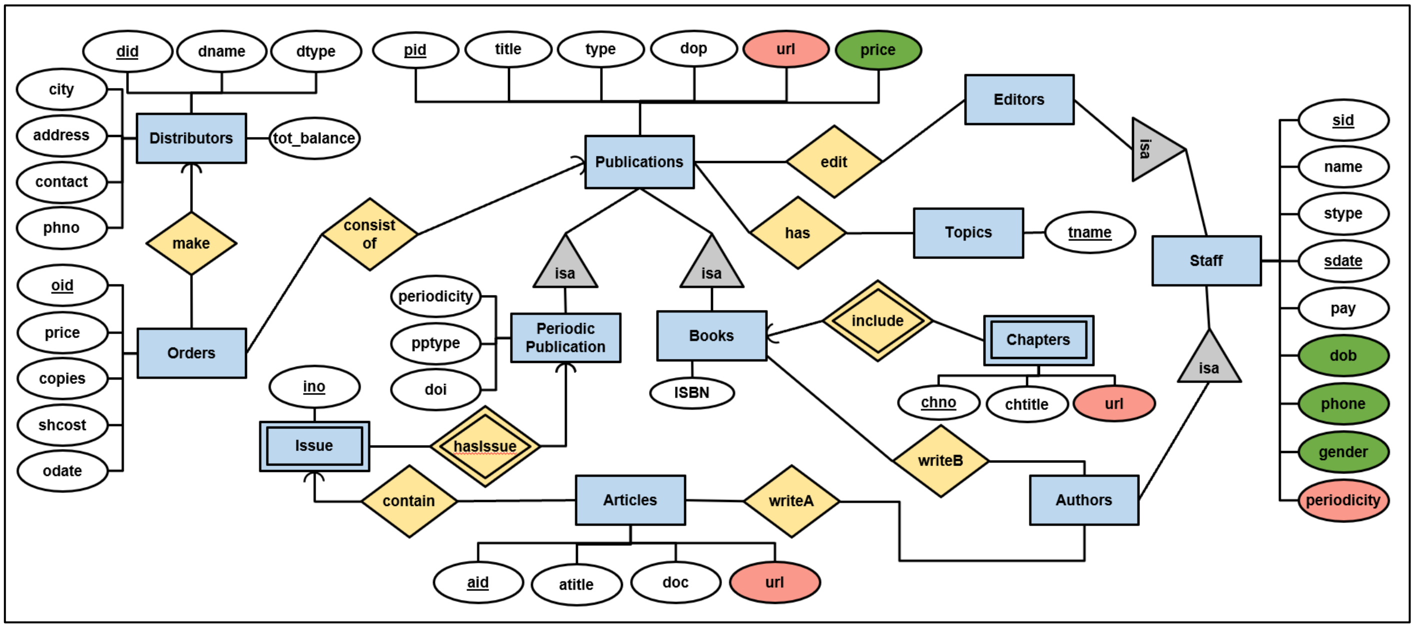
Editors have a rather wide view of the system. They can see details of articles, books, and the authors who wrote them. They also have access to the issues of each publication and the associated topics. However, they do not see the distribution mechanism, i.e., the orders and the distributors.

Authors

Authors have the same view as editors, with restricted access. They can view their articles and books, and modify them, but the rest of the system is read-only. They can only view the publications their articles are a part of, and the editors editing and collating their articles and books.

7. Local E/R Diagram

Mangers' View:

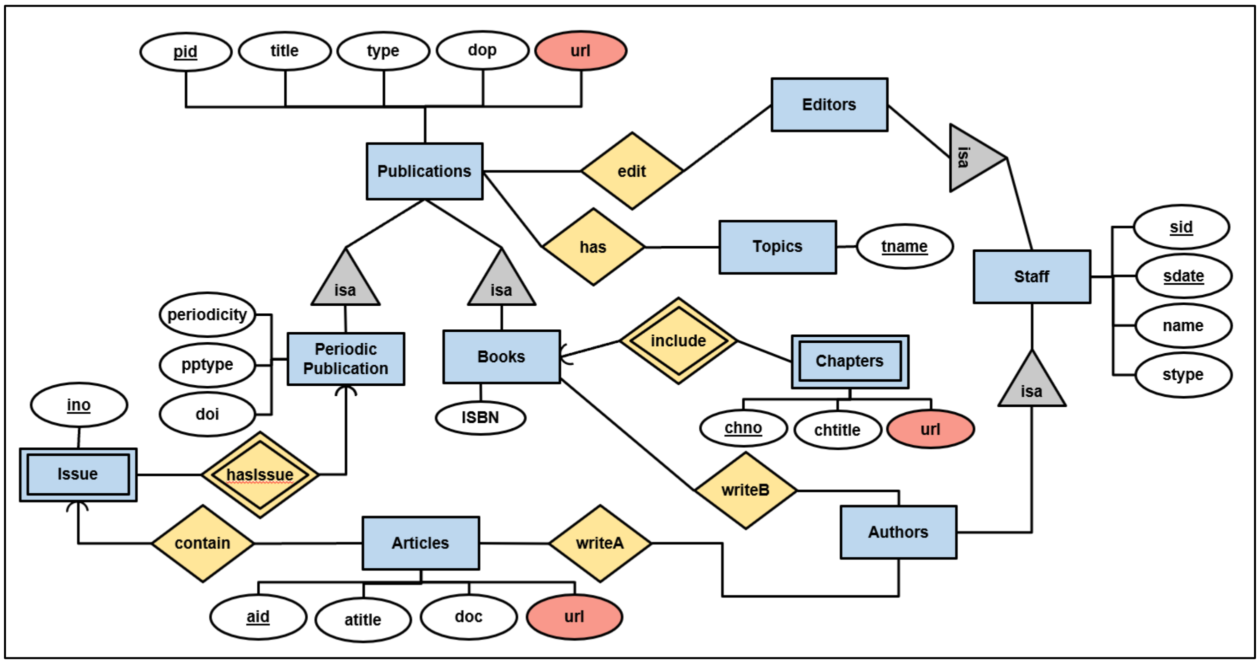


Legend

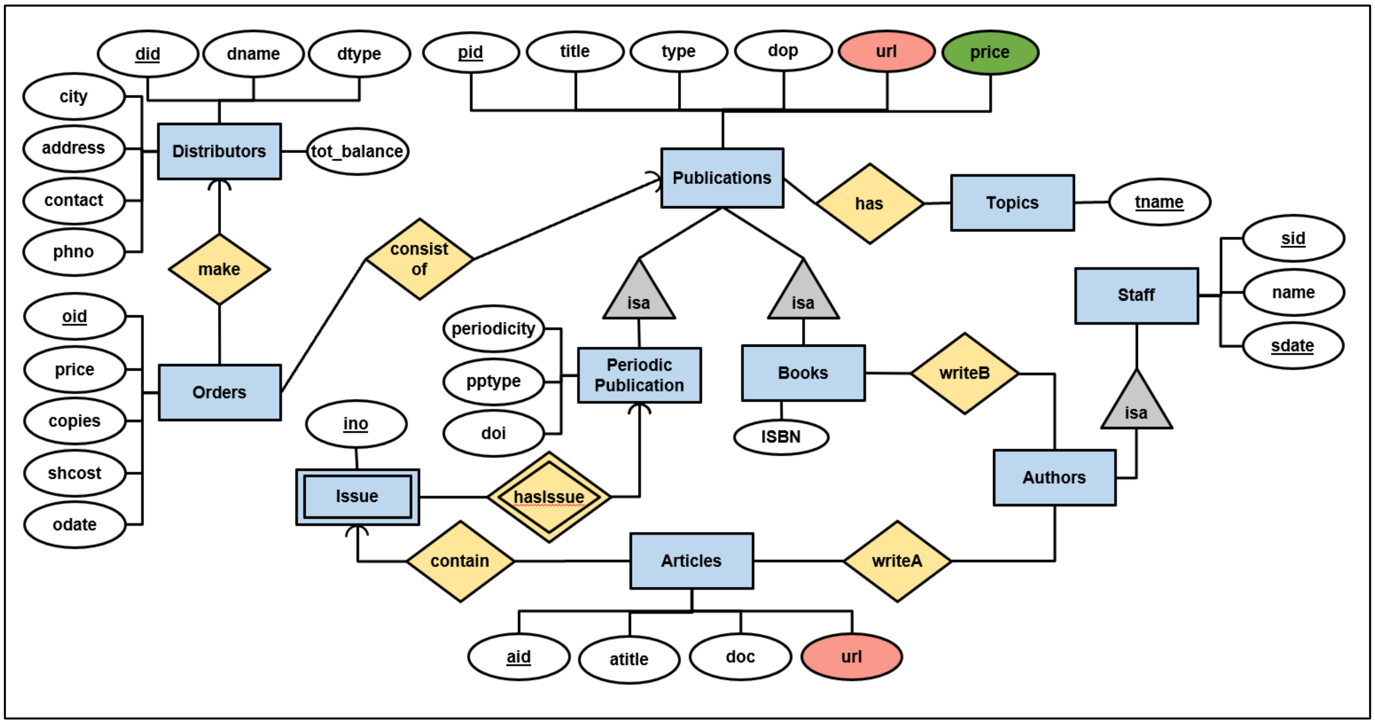
🔴 assumption

🟢 additional information

Editors'/Authors' View:



Distributors' View:



8. E/R Documentation

The following entity sets were created:

1. **Staff** – this entity set stores all information about *WolfCity* publishing house's employees.
2. **Authors** – this entity set was created to define author of each article or book. It is a subclass of Staff entity set, because it shares common attributes with Staff, but participates in certain relationships that other Staff don't.
3. **Editors** – this entity set was created to record editor of each publication. It is a subclass of Staff entity set as it shares common attributes with Staff but participates in certain relationships that other Staff members don't.
4. **Publications** – this entity set stores all information about any publication (books, periodic publication).
5. **Periodic Publications** – this entity set includes different types of periodic publications such as magazine and journals. It is a subclass of Publications entity set.
6. **Books** – this entity set was created to maintain relationship between Publications and Authors. It is a subclass of Publications as it shares common attributes with Publications but participates in different relationships.
7. **Topics** – this entity set stores unique names of topics of each publication. We didn't make it as an attribute, because one publication may have multiple topics.
8. **Orders** – this entity set records all orders made by distributors. Each order corresponds to (possibly multiple copies of) exactly one publication.
9. **Distributors** – this entity set stores all information about Distributors. Distributors are uniquely identified by Distributor ID (`did`).
10. **Issue** – this weak entity set represents the issue number of a periodic publication. Issue number is unique within a periodic publication but can be the same in different periodic publications.
11. **Chapters** – this weak entity set represents each chapter that has unique chapter number (`chno`) within a particular book. It was created in order to provide authors and editors with an access to specific chapters easily via url.
12. **Articles –** this entity set contains all information about articles in the periodic publications.

The following relationships are maintained:

1. **Edit** – Many-to-many relationship as one publication can be edited by many editors and one editor can edit many publications.
2. **Has** – Many-to-many relationship as one publication can have many topics and one topic can be in many publications.
3. **Include** – supporting(weak) many-to-one relationship as each chapter of a book requires PID (publication id) to be uniquely defined.
4. **Writes book** – Many-to-many relationship as one author can write many books and one book can be written by many authors.
5. **Writes article** – Many-to-many relationship as one author can write many articles and one article can be written by many authors.

6. **Contain** – Many-to-one relationship as one periodic publication may contain many articles but one article can be included into exactly one publication.
7. **Has issue** – Relationship between issue and periodic publication. supporting(weak) many-to-one relationship as each issue of periodic publication requires PID to be uniquely defined.
8. **Consist of** – Many-to-one relationship as each order can contain exactly one publication and each publication can be in many orders.
9. **Make** – Relationship between distributors and orders. Many-to-one relationship as each distributor can make many orders but one order can be made by exactly one distributor since each order is defined by unique order_id

9. Local E/R schemas

Manager

```
Staff (sid, name, pay, stype, phone, sdate, pay, periodicity, dob, gender)
Authors (sid)
Editors (sid)
Topics (name)
Publications (pid, title, type, dop, url, price)
Books (pid, ISBN)
Chapters (pid, chno, chtitle, url)
Articles (aid, atitle, doc, url)
Periodic Publication (pid, periodicity, pptype, doi)
Issue (pid, ino)
Orders (oid, price, copies, shcost, odate)
Distributors (did, dname, dtype, tot_balance, phno, city, address, contact)
Make (did, oid)
ConsistOf (oid, pid)
Contain (pid, aid)
Edit (pid, sid)
WriteA (aid, sid)
WriteB (pid, sid)
Has (pid, topic)
```

Editor/Author

```
Staff (sid, name, stype)
Authors (sid)
Editors (sid)
Topics (tname)
Publications (pid, title, type, dop, url)
Books (pid, ISBN)
Chapters (pid, chno, chtitle, url)
Articles (aid, atitle, doc, url)
Periodic Publication (pid, periodicity, pptype, doi)
Issue (pid, ino)
Contain (pid, aid)
Edit (pid, sid)
WriteA (aid, sid)
WriteB (pid, sid)
Has (pid, topic)
```

Distributor

```
Staff(sid, name)
Authors(sid)
Publications (pid, title, type, dop, url, price)
Books (pid, ISBN)
Articles (aid, atitle, doc, url)
Periodic Publication (pid, periodicity, pptype, doi)
```

Issue (pid, ino)
Orders (oid, price, copies, shcost, odate)
Distributors (did, dname, dtype, tot_balance, phno, city, address, contact)
Make (did, oid)
ConsistOf (oid, pid)
Contain (pid, aid)
WriteA (aid, sid)
WriteB (pid, sid)

10. Local Schema documentation

Entity set to relation:

The entity sets in our E/R diagram were converted to relation schemas, with the same attributes. Staff, Publication, Distributor and Order were converted to relation schema and their attributes were same as the E/R diagram.

The Entity sets that inherit 'ISA' of Staff and Publication were made into relation schema based on our E/R diagram. This was done to avoid redundancy.

Relationships to Relation schema

The relationship between the entity set were converted to relation schema. Relationship between entity sets, such as 'edit', 'consist of', 'make', 'has', 'contain', 'writeA', 'writeB' have been converted to a relation schema.

The relationships involving weak entity set, were converted to relation schema as well. Relationship such as 'include' and 'has' been converted into relation schema.