A database design proposal for Hogwarts Library

Mary Miller
December 2nd

Table of Contents

Executive summary 3

E-R Diagram 4

People Table 5

Occupations Table 6

Books Table 7

Alive Books Table 8

Regular Books Table 9

Lost Books Table 10

Rented Alive Books Table 11

Rented Regular Books Table 12

DVD Table 13

Rented DVD Table 14

Lost DVD Table 15

Sample data 16

Views 21

Reports 24

Stored Procedures 27

Security 33

Final Notes 35

Executive Summary

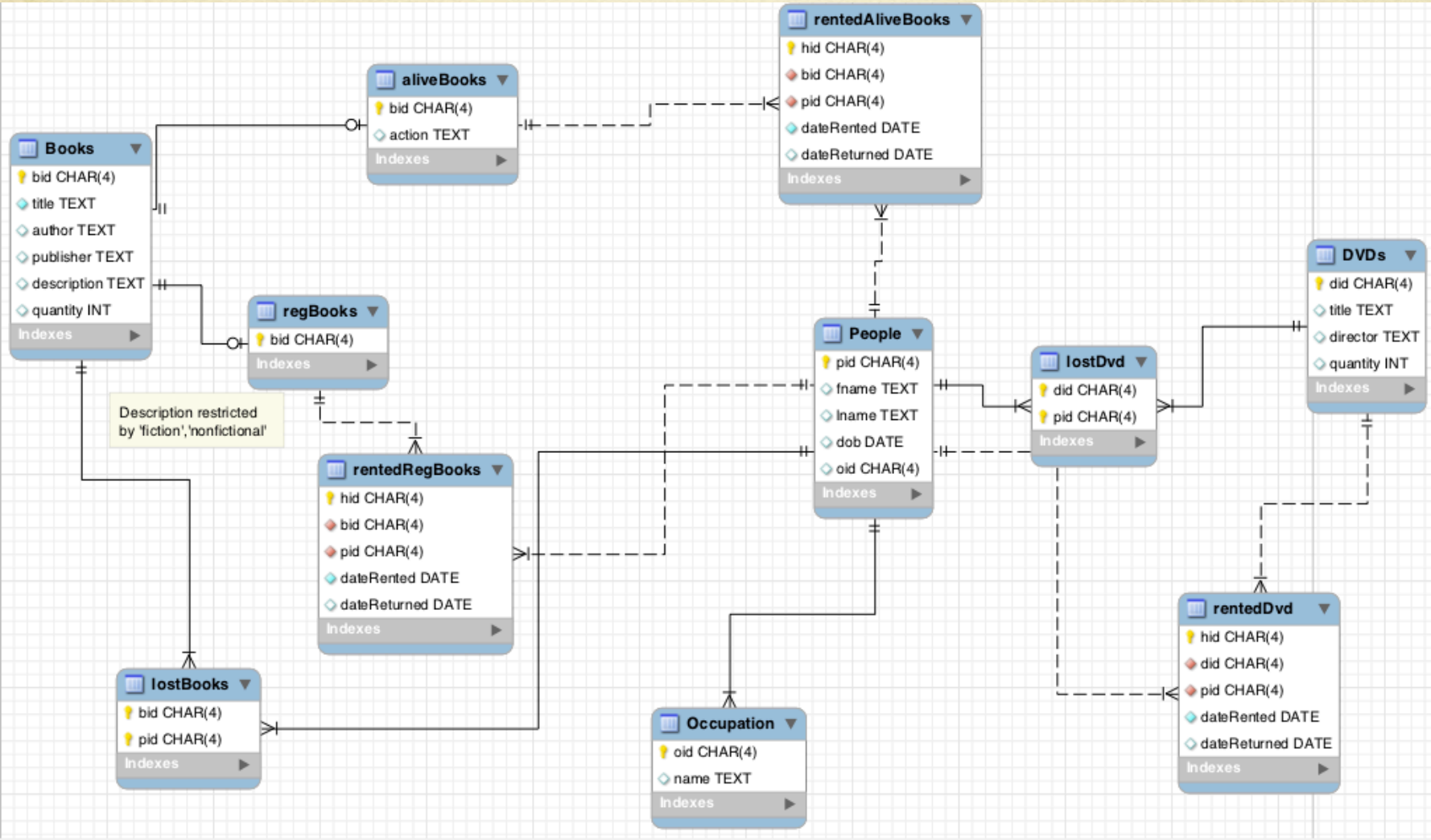
This is a document outlines the database for the Hogwarts library. Hogwarts is home to million's of books and DVD's. These are rented out by students, professors, and other members of the Hogwarts community.

The data being recorded in different tables that include: people, books, occupations, lostbooks, regBooks, aliveBooks, rentedRegBooks, rentedAliveBooks, dvds, lostDvds, rentedDvds. These tables give an overview of what you would be able to find in the hogwarts library. The idea is to allow for easy updates to the system in order to know the history of each book by who rents it out and for how long. Also the same goes for the dvds.

With use of views, reports and stored procedures we are able to see different parts of the database. Exploring these elements will allow for access by different users specified in the securities.

If there was more time permitted there would be more modifications within the database.

Entity Relationship diagram



People Table

This table will hold the people of all members of the Hogwarts community.

FUNCTIONAL DEPENDENCIES

Pid → fname, lname, dob, oid

SQL CREATE STATEMENT

```
DROP TABLE IF EXISTS people;
CREATE TABLE people (
  pid char(4) not null,
  fname TEXT not null,
  lname TEXT not null,
  dob DATE not null,
  oid char(4) not null references occupation(oid),
primary key(pid)
);
```

Occupation Table

The occupation table holds the names of all occupations that could be held at Hogwarts.

FUNCTIONAL DEPENDENCIES

oid → name

SQL CREATE STATEMENT

```
DROP TABLE IF EXISTS occupation;
CREATE TABLE occupation (
  oid char(4) not null,
  name TEXT not null,
primary key(oid)
);
```

Books Table

This table holds all the different books.

FUNCTIONAL DEPENDENCIES

Bid → title, author, publisher, description, inLibrary, quantity

SQL CREATE TABLE

```
DROP TABLE IF EXISTS books;
CREATE TABLE books (
  bid  char(4) not null,
  title TEXT,
  author TEXT,
  publisher TEXT,
  description TEXT check (description='fiction' or description='nonfiction'),
  quantity INT not null,
primary key(bid)
);
```

Alive Books Table

aliveBooks table holds the alive book's bids.

FUNCTIONAL DEPENDENCIES

bid → action

SQL CREATE STATEMENT

```
DROP TABLE IF EXISTS aliveBooks;
CREATE TABLE aliveBooks (
  bid char(4) not null references Books(bid),
  action TEXT,
primary key(bid)
);
```

Regular Book Table

regBooks table holds all bid for regular books.

FUNCTIONAL DEPENDENCIES

bid →

SQL CREATE STATEMENT

```
DROP TABLE IF EXISTS regBooks;
CREATE TABLE regBooks (
  bid char(4) not null references Books(bid),
primary key(bid)
);
```

Lost Books Table

This table holds the lost books and the person to declare it lost.

FUNCTIONAL DEPENDENCIES

bid, pid →

SQL CREATE STATEMENT

```
DROP TABLE IF EXISTS lostBooks;
CREATE TABLE lostBooks (
  bid char(4) not null references Books(bid),
  pid char(4) not null references people(pid),
primary key(bid,pid)
);
```

Rented Alive Books Table

This table keeps track of all rented books that are alive. Hid is the primary key which stands for history id. Allowing for all history alive books a person rents out is recorded.

FUNCTIONAL DEPENDENCIES

hid → pid, bid, dateRented, dateReturned

SQL CREATE STATEMENT

```
DROP TABLE IF EXISTS rentedAliveBooks;
CREATE TABLE rentedAliveBooks (
  hid char(4) not null,
  bid char(4) not null references aliveBooks(bid),
  pid char(4) not null references people(pid),
  dateRented DATE not null,
  dateReturned DATE,
  check (dateRented <= dateReturned),
primary key(hid)
);
```

Rented Regular Books

This table holds the people who are renting out regular books. The primary key is hid which stands for history id. This allows for the Regular books and the people who rented them out to be recorded.

FUNCTIONAL DEPENDENCIES

hid → pid, bid, dateRented, dateReturned

SQL CREATE STATEMENT

```
DROP TABLE IF EXISTS rentedRegBooks;
CREATE TABLE rentedRegBooks (
  hid char(4) not null,
  bid char(4) not null references regBooks(bid),
  pid char(4) not null references people(pid),
  dateRented DATE not null,
  dateReturned DATE,
  check (dateRented <= dateReturned),
primary key(hid)
);
```

DVDs Table

Dvds Table holds all dvds located in the Hogwarts Library and the properties that correspond with it.

Functional Dependencies

did → title, director

SQL CREATE STATEMENT

```
-- DVDs --
DROP TABLE IF EXISTS dvds;
CREATE TABLE dvds (
  did char(4) not null,
  title TEXT not null,
  director TEXT not null,
  quantity INT not null,
primary key(did)
);
```

Rented DVDs Table

rentedDvd table holds all the different dvds and the people who checked them out. It also records what the date was when the dvd was rented and returned.

FUNCTIONAL DEPENDENCIES

hid → did, pid, dateRented, dateReturned

SQL CREATE STATEMENT

```
DROP TABLE IF EXISTS rentedDvds;
CREATE TABLE rentedDvds (
  hid char(4) not null,
  did char(4) not null references dvds(did),
  pid char(4) not null references people(pid),
  dateRented DATE not null,
  dateReturned DATE,
  check (dateRented <= dateReturned),
primary key(hid)
);
```

Lost DVD Table

lostDVD table will hold all the reported lost dvds and the person who last rented it out.

FUNCTIONAL DEPENDENCIES

pid, did →

SQL CREATE STATEMENT

```
DROP TABLE IF EXISTS lostDvds;
CREATE TABLE lostDvds (
  did char(4) not null references dvds(did),
  pid char(4) not null references people(pid),
primary key(did,pid)
);
```

Sample Data

People Table

	pid character(4)	fname text	lname text	dob date	oid character(4)
1	p001	Mark	Mason	1994-10-15	o001
2	p002	Jennifer	Stranson	1995-12-07	o001
3	p003	Alan	Labouseur	1990-02-12	o002
4	p004	Bob	Nisco	1994-09-17	o001
5	p005	Kate	Thompson	1992-02-01	o001
6	p006	Stephanie	Smith	1970-09-29	o002
7	p007	India	Lepoard	1980-11-02	o002
8	p008	Jack	James	1955-01-17	o003

Book Table

	bid character(4)	title text	author text	publisher text	description text	quantity integer
1	b001	Wonderful World of Hogwarts	Dumbledore	Witch Digest	nonfiction	3
2	b002	Dark Arts for all ages	Kathy Pinpimple	Magicial Cycle	nonfiction	2
3	b003	The Lost Witch	Angus McSnuffles	SNAS	fiction	2
4	b004	Muggles lost in diagon alley	James Matches	Children CO.	fiction	1
5	b005	Beginning of Magic	Lady Lollipop	Maches Matience	nonfiction	5

Sample Data cont.

Occupation Table

	oid character(4)	name text
1	o001	Student
2	o002	Professor
3	o003	Janitor

aliveBook Table

	bid character(4)	action text
1	b003	Flying Witch
2	b001	Animated Pictures

regBoook Table

	bid character(4)
1	b002
2	b004
3	b005

Sample Data cont.

rentedAliveBook Table

	hid character(4)	bid character(4)	pid character(4)	daterented date	datereturned date
1	h001	b001	p001	2013-09-09	2013-09-16
2	h002	b001	p002	2013-09-11	
3	h003	b001	p002	2013-10-01	2013-11-10
4	h004	b003	p006	2013-08-15	2013-08-20
5	h005	b001	p005	2013-11-25	
6	h006	b003	p007	2013-11-27	
7	h007	b001	p008	2013-11-26	
8	h008	b003	p001	2013-11-25	

Sample Data cont.

rentedRegBook Table

	hid character(4)	bid character(4)	pid character(4)	daterented date	datereturned date
1	h001	b004	p001	2013-09-01	2013-09-18
2	h002	b004	p007	2013-10-10	
3	h003	b005	p004	2013-11-10	2013-11-17
4	h004	b002	p008	2013-11-10	2013-12-01
5	h005	b004	p007	2013-11-15	2013-12-01
6	h006	b002	p002	2013-11-27	
7	h007	b005	p003	2013-11-29	
8	h008	b005	p006	2013-12-01	
9	h009	b002	p005	2013-11-29	

DVDs Table

	did character(4)	title text	director text	quantity integer
1	d001	Magic 101	Corning Corny	1
2	d002	How to Kill in 10 easy steps	Magic Hanson	2
3	d003	The one who shall not be named	Lou Kingston	1
4	d004	Poition that up	Isibell Mickel	1

Sample Data cont.

lostDVDs Table

	did character(4)	pid character(4)
1	d002	p005
2	d004	p001

rentedDvds Table

	hid character(4)	did character(4)	pid character(4)	daterented date	datereturned date
1	h001	d002	p008	2013-10-09	2013-10-16
2	h002	d004	p001	2013-10-10	
3	h003	d001	p003	2013-11-08	2013-11-13
4	h004	d002	p005	2013-11-23	2013-11-29

Views

PeopleOccupationReg allows for the regular books and it's association with people and occupation. This view could be used by the librarian to see who has rented out the books.

SQL VIEW CREATE STATEMENT

```
CREATE VIEW PeopleOccupationReg AS
SELECT distinct p.pid as pid,
       p.fname as First_Name,
       p.lname as Last_Name,
       o.name as Occupation,
       b.title as Book_Title
FROM people p,
     occupation o,
     books b,
     rentedRegBooks rrb
WHERE p.oid = o.oid
  and b.bid = rrb.bid
  and p.pid = rrb.pid
ORDER BY p.lname ASC
```

USE EXAMPLE

```
SELECT *
FROM PeopleOccupationReg
WHERE first_name = 'Jack'
```

Views

peopleOccupationAlive allows for the regular books and it's association with people and occupation. This view could be used for librarians to see you took out an alive book and what their occupation is.

SQL VIEW CREATE STATEMENT

```
CREATE VIEW PeopleOccupationAlive AS
SELECT distinct p.pid as pid,
       p.fname as First_Name,
       p.lname as Last_Name,
       o.name as Occupation,
       b.title as Book_Title
FROM people p,
     occupation o,
     books b,
     rentedAliveBooks rab
WHERE p.oid = o.oid
  and b.bid = rab.bid
  and p.pid = rab.pid
ORDER BY p.lname ASC
```

USE EXAMPLE

```
SELECT *
FROM PeopleOccupationAlive
WHERE occupation = 'Student'
```

Views

PeopleOccupationDVD allows for the dvd's and it's association with people and occupation. Allows for the librarian to see the dvds and who has rented out what dvd throughout history.

SQL VIEW CREATE STATEMENT

```
CREATE VIEW PeopleOccupationDVD AS
SELECT distinct p.pid as pid,
       p.fname as First_Name,
       p.lname as Last_Name,
       o.name as Occupation,
       d.title as DVD_Title
FROM people p,
     occupation o,
     dvds d,
     rentedDvds rd
WHERE p.oid = o.oid
  and p.pid = rd.pid
  and d.did = rd.did
ORDER BY p.lname ASC
```

USE EXAMPLE

```
SELECT *
FROM PeopleOccupationDVD
WHERE dvd_title = 'How to Kill in 10 easy steps'
```

Reports

These Reports can be used by librarians to check on books and who has rented them out. This could be displayed in a web interface or some sort of graphical interface that the hogwarts library chooses.

Frist report results in how many dvds each person has rented out over history.

SQL REPORT CREATE STATEMENT

```
SELECT p.fname,
       p.lname,
       count(rd.did)
From People p,
     Dvds d,
     rentedDvds rd
WHERE p.pid = rd.pid
  and d.did = rd.did
group by p.fname, p.lname
```

Reports

This report explains how many regular books each person has rented out.

SQL REPORT CREATE STATEMENT

```
SELECT p.fname,
       p.lname,
       count(rrb.bid)
From People p,
     books b,
     rentedRegBooks rrb
WHERE p.pid = rrb.pid
  and b.bid = rrb.bid
group by p.fname, p.lname
```

Reports

This report executes how many alive books each person has rented out throughout history.

SQL REPORT CREATE STATEMENT

```
SELECT p.fname,
       p.lname,
       count(rab.bid)
From People p,
     books b,
     rentedAliveBooks rab
WHERE p.pid = rab.pid
  and b.bid = rab.bid
group by p.fname, p.lname
```

Stored Procedures

List all people who have rented the specific alive book from the library

SQL STORED PROCEDURES CREATE STATEMENT

```
CREATE FUNCTION listAliveBookPeople(bookTitle text)
RETURNS TABLE(People_Fname text, People_lname text, occupation text) AS $$
BEGIN
        RETURN QUERY SElECT p.fname,p.lname,o.name
                     FROM people p,
                          occupation o,
                          books b,
                          rentedAliveBooks rab
                     WHERE b.title = bookTitle
                       and o.oid = p.oid
                       and rab.pid = p.pid
                       and rab.bid = b.bid;
END;
$$ LANGUAGE PLPGSQL
```

USE EXAMPLE

```
SELECT listAliveBookPeople('Wonderful World of Hogwarts')
```

Stored Procedures

List every person who has rented out a regular book and their occupation.

SQL STORED PROCEDURES CREATE STATEMENT

```
CREATE FUNCTION listRegularBookPeople(bookTitle text)
RETURNS TABLE(People_Fname text, Student_lname text, occupation text) AS $$
BEGIN
        RETURN QUERY SElECT p.fname,p.lname,o.name
                     FROM people p,
                          occupation o,
                          books b,
                          rentedRegBooks rrb
                     WHERE b.title = bookTitle
                       and o.oid = p.oid
                       and rrb.pid = p.pid
                       and rrb.bid = b.bid;
END;
$$ LANGUAGE PLPGSQL
```

USE EXAMPLE

```
SELECT listRegularBookPeople('Beginning of Magic')
```

Stored Procedures

This stored procedure shows a list of all Regular books that this person rented out.

SQL STORED PROCEDURE CREATE STATEMENT

```
CREATE FUNCTION peopleRegularListBooks(studentFName text, studentLName text)
RETURNS TABLE(BookTitle text, author text) AS $$
BEGIN
        RETURN QUERY SElECT b.title, b.author
                     FROM people p,
                          books b,
                          rentedRegBooks rrb
                     WHERE p.fname = studentFName
                       and p.lname = studentLName
                       and rrb.pid = p.pid
                       and rrb.bid = b.bid;
END;
$$ LANGUAGE PLPGSQL
```

USE EXAMPLE

```
SELECT peopleRegularListBooks('Bob','Nisco')
```

Stored Procedures

This returns a list of all alive books the specified person rented out.

SQL STORED PROCEDURES CREATE STATEMENT

```
CREATE FUNCTION PeopleAliveListBooks(studentFName text, studentLName text)
RETURNS TABLE(BookTitle text, author text) AS $$
BEGIN
        RETURN QUERY SElECT b.title, b.author
                     FROM people p,
                          books b,
                          rentedAliveBooks rab
                     WHERE p.fname = studentFName
                       and p.lname = studentLName
                       and rab.pid = p.pid
                       and rab.bid = b.bid;
END;
$$ LANGUAGE PLPGSQL
```

USE EXAMPLE

```
SELECT PeopleAliveListBooks('Stephanie','Smith')
```

Stored Procedures

List People who have rented out the dvd specified.

SQL STORED PROCEDURES CREATE STATEMENT

```
CREATE FUNCTION listDVDPeople(dvdTitle text)
RETURNS TABLE(Student_Fname text, Student_lname text, occupation text) AS $$
BEGIN
        RETURN QUERY SELECT p.fname,p.lname,o.name
                     FROM people p,
                          occupation o,
                          dvds d,
                          rentedDvds rd
                     WHERE d.title = dvdTitle
                       and o.oid = p.oid
                       and rd.pid = p.pid
                       and d.did = rd.did;
END;
$$ LANGUAGE PLPGSQL
```

USE EXAMPLE

```
SELECT listDVDPeople('Magic 101')
```

Stored Procedures

Given a first name and a last name, a list of dvds that the specified person has rented out.

SQL STORED PROCEDURES CREATE STATEMENT

```
CREATE FUNCTION peopleDVDList(studentFName text, studentLName text)
RETURNS TABLE(DvdTitle text, director text) AS $$
BEGIN
        RETURN QUERY SElECT d.title, d.director
                     FROM people p,
                          dvds d,
                          rentedDvds rd
                     WHERE p.fname = studentFName
                       and p.lname = studentLName
                       and rd.did = d.did
                       and rd.pid = p.pid;
END;
$$ LANGUAGE PLPGSQL
```

USE EXAMPLE

```
SELECT peopleDVDList('Jack','James')
```

Security

Security will be broken down to permissions based on table.

People Table

The contents of the people table will only be able to be viewed by the individual person.

Book Table

The book table will only be able to be accessible by librarians to modify. People will be able to see all the books in the Hogwarts Library

Occupation Table

This table would only be able to accessed by an administration.

Lost Books Table

Lost books table will be accessible by the librarians only

RegBooks Table

The information in this table will only be able to be accessed by the Librarians.

Security cont.

Alive Books Table

Only accessible by the librarians.

Rented Regular Books Table

This table in full will be able to be accessed by the Librians and there will be restrictions to what a person can see. They can see that it is rented out to someone else however, they cannot see who specifically has it.

Rented Alive Books Table

This table is the same as **Rented Regular Books Table** restrictions.

DVDs Table

This table will be accessible by the librarians for changes. People will be able to see the Dvds that Hogwarts has.

Rented Dvds Table

Will be able to be edited by the Librarians, and people will be able to view if a DVD is being rented.

Lost Dvds Table

This table will only be able to be edited by the Librarian.

Final Notes

Known Problems within this design of the Hogwarts library

- Separation of alive and regular books causes for difficulty to bring an entire list of values together.

Future enhancements that could be made are:

- Add more fields to the tables
- Add in Newspapers table and other items found within a library.
- Add in a librarian table and/or Administration Table

For Source code please see SQL code on github if you want