Final Database Design Deliverables

CS x265, Introduction to Database Management Systems, Spring 2018

Douglas H. Fisher

Each group will submit three documents to Brightspace by **Friday, March 2 at 6:00 pm**.

1. Submit a pdf **group affirmation**, which must be signed, scanned, and uploaded.

2. The **Complete Implementation File** is an SQLite .sql (or .txt) file with all CREATE TABLE, CREATE TRIGGER, and CREATE VIEW statements, as well as INSERT statements to populate the database, and queries (SELECT statements).

3. The **Complete Design Document**

Each of these items is described in greater detail below.

1. Group Affirmation

Each team member should sign the first statement. If an individual can't honestly sign such a statement, then don't sign it, but we will talk.

One or Two Lines

Giving the Title of your Project

Group <your group #>

Each group member's name, in 1-4 lines (12 pt)

Statement 1: "*We have each contributed to this design and design document, and though we may have divided some work up between team members, we have checked each others work, we have each checked the totality of the design, and we have sought to synthesize our contributions into one cohesive, robust, and elegant design.*"

Each group member signs here

------------------------------------- End of group affirmation and release form -------------------------------

2. Complete Implementation File

You will submit an **SQLite .sql (or .txt) file** with CREATE TABLE, CREATE TRIGGER, and CREATE VIEW, as well as INSERT statements to build a sample database, and queries to test the implementation. The beginning of your file should be formatted much like the Widom course sample .sql files, such as rating.sql, a snapshot of which is shown here:

```
/* GIVE the GROUP number and GROUP member names who did any coding – I hope all */

    /* Delete the tables if they already exist */
    drop table if exists Movie;
    …
    /* Create the schema for our tables */
    create table Movie(mID int, title text, year int, director text);
    …
    /* Populate the tables with our data */
    insert into Movie values(101, 'Gone with the Wind', 1939, 'Victor Fleming');
    insert into Movie values(102, 'Star Wars', 1977, 'George Lucas');
    …
    insert into Reviewer values(201, 'Sarah Martinez');
    insert into Reviewer values(202, 'Daniel Lewis');
    …
    insert into Rating values(201, 101, 2, '2011-01-22');
    insert into Rating values(201, 101, 4, '2011-01-27');
    …
```

a) Presumably, your table definitions, in the form of CREATE TABLE statements, will be much richer than Widom's, with in-table checks, primary keys, foreign key constraints, and the like.

- In your table definitions **do NOT assume defaults for ON DELETE and ON UPDATE** clauses for FOREIGN KEYs. Comment on choices that you think are interesting, and questions on interactions between ON DELETE and ON UPDATE actions are candidates for the exams, so understand my previous examples.

b) The various initial INSERT statements, presumably using the VALUES format, in this file is where I will find your **sample data**.

c) All CREATE and SELECT statements and queries should be preceded by header comments /* … */, which will appear in the SQLite .SQL file immediately above the corresponding construct, explaining (for example) why you chose one translation strategy over another (e.g., you might point to the UML in doing so), why you declared attributes to be of particular types, and other information that might be helpful in understanding how your implementation realizes your vision.

d) In header comments for each CREATE (TABLE, VIEW, TRIGGER) statement, list the primary author(s) of the statement, and those team members who reviewed the statement for correctness and consistency with the UML (see Complete Design Document).

e) Place DROP TABLE, DROP VIEW, and DROP TRIGGER at the START and END of your file, so that upon exit, our environment will be cleared for the next group. DROP statements at both the start and end of file may seem a bit much, but it can ease grading if there are errors in one group's file that precede another group's file.

f) Each team member should define at least one view over multiple base tables. Motivate and explain each of these views, with at least part of each rationale being related to security, and the principle of "need to know" (e.g., a "user" only needs access to information concerning them). You may, of course, define more than one view per group member, but I am really more interested in a small number of sophisticated views (e.g., see the multi-table example in the in-class exercise on Views), rather than seeing a bunch of very simple, one-table views. **Each view should have at least one primary author among the group members, and again each member must be the primary author of at least one view** (though all group members should review each view). **Again, in header comments of each view, identify the primary author and the reviewers.**

g) Each team member should define at least one trigger, which can be INSTEAD OF triggers to implement operations on views, and/or BEFORE/AFTER triggers to approximately or exactly implement constraints dictated by assertions, and/or other. Similar to views, **each trigger should have a primary author, and each member must be the primary author of at least one trigger** (though all group members should review each trigger). **Again, in header comments of each trigger, identify the primary author and the reviewers.**

h) In addition, you will include a series of **at least 10 queries (SELECT)** that you used to test and demo your database implementation. Its possible that you will include the same query twice in illustrating some aspect of the DB, but just count this once as part of your "at least 10".

- Also, **at least five queries should involve GROUP BY, HAVING BY, and aggregation**. Each group member should be primary author of one of these queries, with the primary author and reviewers given in header comments.

i) In comments at the bottom of the implementation file, list the results of running each of the 10+ queries on the database.

j) In writing your statements, use a convention of CAPITALIZING all SQL reserved words (e.g., CREATE), and **use indentation as necessary and that is pleasing to your eye** (chances are, it will be pleasing to others as well, notably me and graders) when read on your formatted pages.

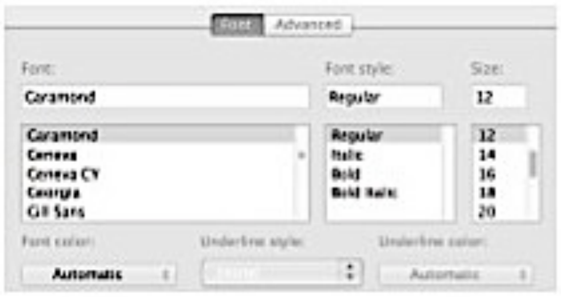
k) Note that you also will be appending a copy of the implementation .sql file to the complete design document (as Section 4).

3. Complete Design Document

There are several sections of the design document. All textual components of this document (except the UML) will use Garamond font, 12 pt font; 1 inch margins; single line spacing (except as otherwise noted). You will be uploading it as one, large PDF document to Brightspace

COVER PAGE

At the top of the Cover page, give the title of your project in bold-face 14 pt, and **give your Project Group number** in bold face 12 pt font (much like the affirmation and release form), and **your group members' names.**



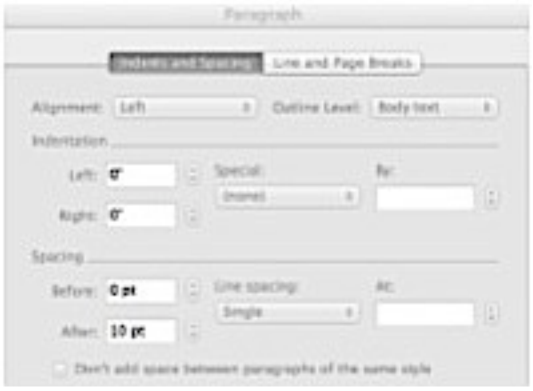
Font: Garamond
Font style: Regular
Size: 12

Garamond
Geneva
Geneva CY
Georgia
Gill Sans

Regular
Italic
Bold
Bold Italic

12
14
16
18
20

Font color: Automatic
Underline style:
Underline color: Automatic



Alignment: Left
Outline Level: Body text

Indentation
Left: 0"
Right: 0"
Special: (none)
By:

Spacing
Before: 0 pt
After: 10 pt
Line spacing: Single
At:

Don't add space between paragraphs of the same style

You may add a picture, perhaps a collage that exemplifies or illustrates your project, in the middle of the cover page. Hopefully your picture will be better than my reduced screenshot, which illustrates format settings in Word. **The images in your picture should all be licensed for use. Acknowledge the image sources** in a footnote that appears on the cover page.

SECTION 1. Overview

Prepare a one-page overview of your design. Include an

- opening paragraph on motivation and vision for the project, then
- the functional specification with major functions in a bulleted list, and then
- describe your DB's major design elements to implement these functions,
    - with pointers into the remainder of the design document as you see fit, most particularly the UML.

SECTION 2. Comprehensive UML diagram of database

a) UML diagrams cannot be hand-drawn, but should be done in PowerPoint (or similar tool that can be used to produce **UML consistent with the class UML conventions**). Use my own UMLs as examples of the neatness standard that I expect (e.g., university records, Book DB, Dorm energy DB).

b) You can place text annotations (e.g., a concise statement of what a class or relation is intended to represent, if it doesn't seem self evident) in the UML, but avoid clutter. If you wish to elaborate on points (e.g., justification for a particular class, relation, or other design choice; on how your UML design realizes your vision), then include Endnotes immediately after your UML to do so, which are numbered and in most cases are referenced from the same numbered "citation" that is placed as annotation in the UML diagram itself (e.g., "See Endnote 3").

c) Use sharp, well-defined lines. If you use a special graphical formatting tool, make sure that it's consistent with our course's variant of UML (if you want an exception, please get it from me ahead of time).

d) Label (i.e., give a name to) any association that will be translated as a separate table using SQL CREATE TABLE statements. When an association involves a 1..1 constraint, I would almost never translate the association into a table, and I would often not translate an association that involves a 0..1 into a separate table either. **Label roles of a self-association when the association is NOT symmetric** (e.g., PreReqOf in a university database). In contrast, when a self-association is symmetric (e.g., CourseEquiv in a university database) the attribute names in the translated tables will be somewhat arbitrary (e.g., Course1, Course2) and there is no need to label them in the UML.

e) Some groups may have to give their UML's in parts, on different pages. For example, Figure 1 gives the full definitions and context of these tables in a university database

- Course
- Section
- Participant
- Instructor
- Student
- Term

All attributes of these six classes would be listed, had I bothered to list them, in the respective Class boxes of Figure 1. In addition, all associations (including association classes) in which these six classes participate are fully defined in Figure 1, including cardinality constraints on both ends of the association. In short, I can look at this one page for the "full" definition of these six classes and their various associations, but I must look on other pages to find the definitions of other classes, three of which are given as stubs in Figure 1, with an indication of where their full definitions can be found. So Resource, Recommendation, and Program are stubs, and I must look in Figures 2 and 3 for their full definitions.

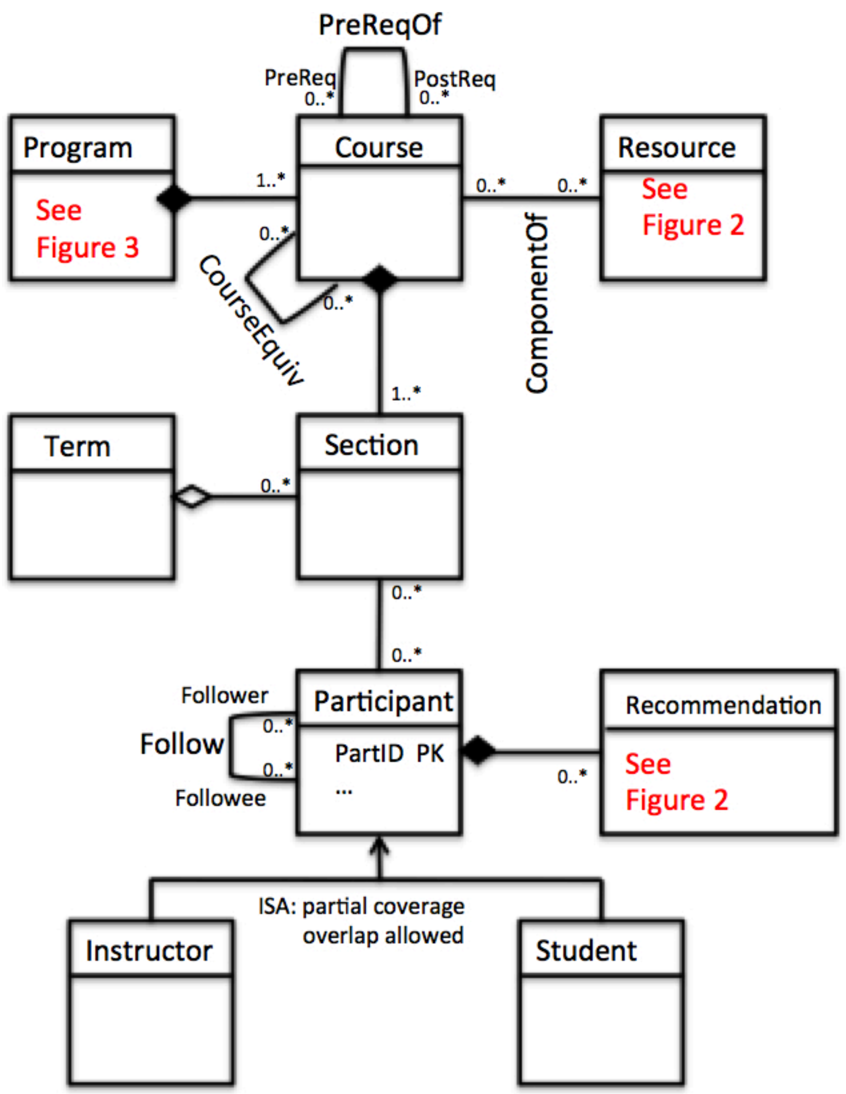


Figure 1: partial UML of student/instructor portfolios

Figure 2 shows the complete definitions for Resource and Recommendation (and again, the class boxes would have listed all attributes for those classes in Figure 2). Pointers back to Figure 1 are given for Course and Participant, and the associations are stripped of all specificity, because that is given in Figure 1 as well. The idea is to eliminate as much redundancy (and therefore potential for inconsistency) as possible, while making the UML unambiguous, even though it is split across multiple pages.

Similarly, Figure 3 shows the full definition of class Program (other than specifics of associations that have been defined elsewhere), referenced from Figure 1. In addition, class Institution makes its first appearance and is fully defined in Figure 3.

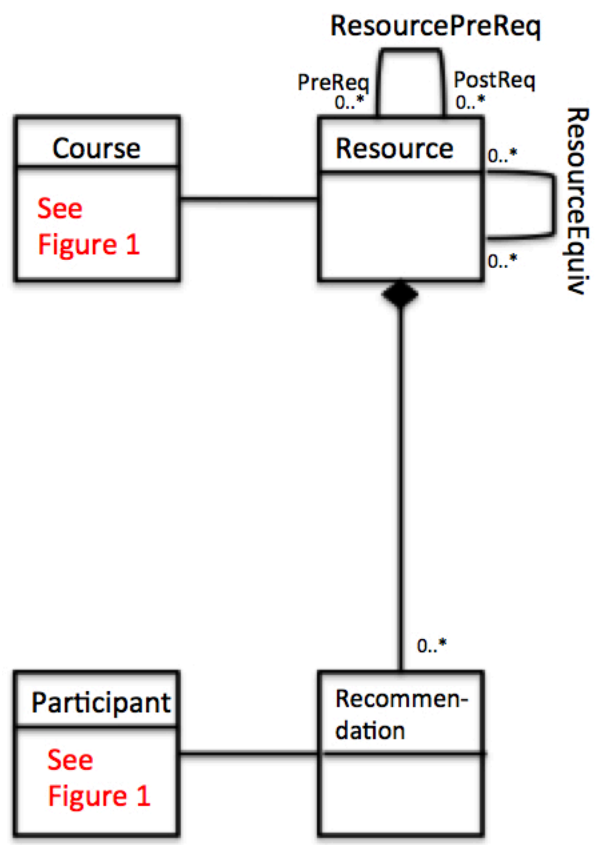


Figure 2: more partial UML of student/instructor portfolios

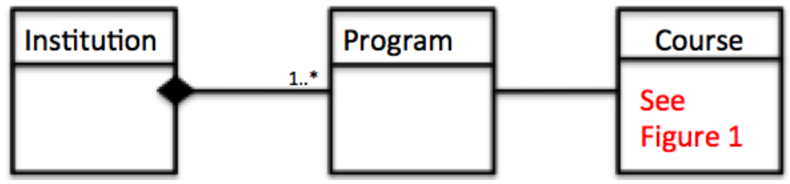


Figure 3: final partial UML of student/instructor portfolios

SECTION 3. General assertions

List any assertions that you developed as part of the design. Include assertions that represent **1..* cardinality (aka participation or multiplicity) constraints, disjoint (no overlap) across subclasses, and complete coverage of subclasses. Include other assertions as appropriate to be consistent with the UML.**

Give the authors and reviewers of each trigger in header comments.

For each of the assertions that you end up specifying in this section, I want you to also point out how you enforced the constraints implied by the assertion in the final database design. For example, consider this assertion from the Books database that you were given in earlier classes, which ensure that each book (Isbn) participates in association with at least one author through WrittenBy.

```
CREATE ASSERTION BookWrittenByConstraint  /* ensure 1..* constraint of Books in */
CHECK (NOT EXISTS (SELECT *               /* WrittenBy. Other constructs possible. */
        FROM Book B
        WHERE B.Isbn NOT IN (SELECT W.Isbn FROM WrittenBy W)))
```

In most cases you could enforce constraints represented by general assertions with TRIGGERs. In theory, general assertions could be approximated by in-table CHECKs, but since SQLite (or any platform) doesn't support in-table CHECKs with nested queries, you could not use in-table CHECKs in most cases.

For each assertion you give in this section, point to the implementation file (also Section 4 below) and explain how you approximated the assertion's affects through other means.

SECTION 4. Implementation file: Comprehensive Table, Inserts, Deletes, Updates, Triggers

Simply list the code from your .sql complete implementation file in this section of your complete design document. Reformat minimally to eliminate undesirable line breaks, and line overflow – make it look presentable in this document. You may use a different font than Garamond for this if you want.

A major component of grading will be to check for consistency between the UML and the SQL definitions, most notably tables, but also triggers. Inconsistencies will carry much greater weight than they did in prior project deliverables.

This section may include additional explanatory text on your reasons for a particular translation strategy, and other matters regarding implementation that you want to elaborate on, such as implementation challenges that you faced.