Final Database Design Deliverables

CS 265, Introduction to Database Management Systems, Spring 2015

Group 0

Douglas H. Fisher

Each group will submit four items to complete Project 2.

1. **Complete Design Document** must be uploaded by **Saturday, April 18 at 11:59 pm**.

2. **Complete Implementation File** is an SQLite .sql file with all CREATE TABLE, TRIGGER, INDEX, VIEW statements, as well as INSERTs, DELETEs, UPDATEs, and queries (SELECT). This must be uploaded by **Saturday, April 18 at 11:59 pm.**

3. **Video presentation** summarizing the project must be available by **Monday, April 20 at 6:00 pm** (private or public) link on Youtube, Box, or some other host.

4. **Group affirmation and release form** must be scanned and uploaded as a PDF by **Monday, April 20 at 6:00 pm.**

Each of these items is described in greater detail below, though not in this order.

Group Affirmation and Release Form

The last document that you submit will be the **Group affirmation and release form**, which will be formatted with title lines in Garamond 14 pt bold face font, and group number and names in 12 pt Garamond bold-face font. Three statements follow the title and team lines.

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! If group members do NOT agree to a policy of the remaining two questions, then I will not make any Project 2 material available on the Web in any form. You can use the following template for the group affirmation and release form.

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.*"

_____ _____ _____ _____

Statement 2: "*We DO / DO NOT (choose one) agree to have our project's written document and video posted to the Web.* " (If you only want to release ONE of the written document or video, then select "DO" and put a line through the option you do NOT want to permit)

_____ _____ _____ _____

Statement 3 Only if you agree above*: "We DO / DO NOT (choose one) want our names attached to the Web-posted document and video on our project.*"

_____ _____ _____ _____

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

Video Presentation

You will prepare a (roughly) 5 minute video, using one of a variety of freely available or at cost tools (e.g., Screenflow, Screenr, Camtasia). Your video should be structured as follows.

> (a) application overview, perhaps opening with a grand vision and summarizing the functional specification, perhaps with illustrations, but minimally with slide deck of bullet items;
>
> (b) an overview of the design, with a focus on the UML, highlighting different parts of the design, and highlighting relationships between the design and the functional specification;
>
> (c) a demo of selected queries on a small sample database; and
>
> (d) a very brief summary of future directions that this project could develop.

You can get as creative as you want on this video, but I expect noting more than a screen capture presentation. I advise scripting it ahead of time, however, and giving each team member some speaking lines (but that is optional). I was impressed at how well the lightening talks went after project 1, and I am looking to capture that on video.

I have created a Blackboard Forum to post about video screen capture tools. Please post if you have favorite tools you'd like to let your colleagues know about.

Complete Implementation File

You will submit an **SQLite .SQL file** with CREATE TABLE, CREATE TRIGGER, CREATE VIEW, and CREATE INDEX statements, as well as INSERT statements to build a sample database. Preface the table with DROP TABLE, DROP VIEW, DROP TRIGGER, and DROP INDEX statements, in case we run through the database setup multiple times. 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 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');
...
```

But presumably, your table definitions will be richer, with in-table checks in some cases, foreign key constraints, and the like. The various initial INSERT statements, using the VALUES format, in this file is where I will find your **sample data**.

Also, include all the **DROP statements at the end of your file as well**, so that upon exit, our environment will be cleared for the next group. If we hang in the middle, we'll start with a new session for the next group.

You should have **at least two triggers**, **at least three views**, and **at least five indexes** (consistent with https://my.vanderbilt.edu/cs265/projects/). See the section on indexes under the design document regarding indexes and their justification. One of the views will be over a single base table, and one that is a view over multiple tables; you get your pick on the third. 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 two views, but I am really more interested in one sophisticated view (e.g., see the multi-table example in the in-class exercise on Views), rather than seeing a bunch of fairly simple views.

In addition, you will include a series of **at least 23 queries (SELECT) and other INSERT (with SELECT statement), DELETE, UPDATE commands** that you used to test and demo your database implementation. I had previously stipulated 25 (but counted the two triggers mentioned above as two of those 25). 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 23". Also, **INSERT statements using VALUES, including those that create your initial database instance, don't count towards these 23.**

Also, consistent with https://my.vanderbilt.edu/cs265/projects/, **at least two queries should involve GROUP BY, HAVING BY, and aggregation**, and **at least two should be correlated queries**. Highlight these different classes of queries in your header comments (e.g., "THIS IS ONE OF OUR CORRELATED QUERIES").

All CREATE 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, **why you chose to index a given attribute (see Appendix)**, and other information that might be helpful in understanding how your implementation realizes your vision.

In your table definitions **do NOT assume defaults for ON DELETE and ON UPDATE** clauses for FOREIGN KEYs. I am not asking you to justify these in writing, but do /* Comment */ on choices that you think are interesting, and questions on interactions between ON DELETE and ON UPDATE actions are candidates for the final exam, so understand my previous examples.

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) when read on your formatted pages.

Note that you also will be appending a copy of the implementation .sql file to the complete design document.

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 Blackboard.

COVER PAGE

At the top of the Cover page, give the title of your project in bold-face 14 pt, and **give your Project 2 Group number** in bold face 12 pt font (much like the affirmation and release form). **Do NOT give your groups names.** Use 10pt spacing between paragraphs.

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 such that you can use, with sources acknowledged** 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
- a 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 (UML)

Below is a sample, albeit incomplete UML -- this UML was just created to illustrate some formatting conventions, and is otherwise incomplete (e.g., lack of attributes in classes).

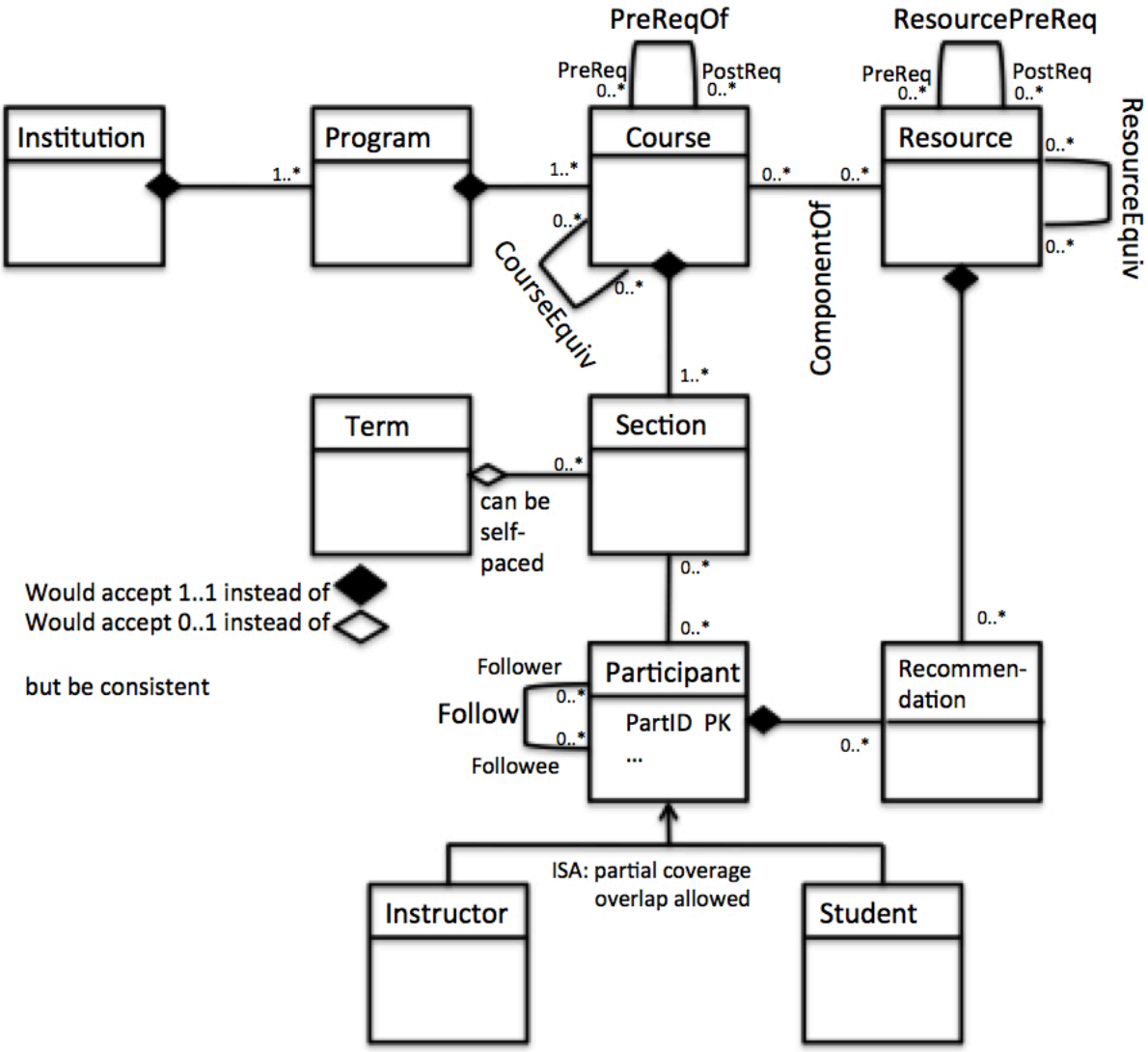

Figure 0: A sample UML DB model (perhaps for maintaining learner and teacher portfolios in an unbundled world).

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. For example, I placed "can be self-paced" to explain the 0..1 cardinality constraint between Term and Section. 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").

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).

Label any association that will be translated as a separate table using SQL CREATE TABLE statements (and that appear in the executable implementation file of the previous section, and in the

final section of this document). You'll note that I labeled such associations in the diagram above, but I did not label associations that would not be translated into separate tables. 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 contrast, when a self-association is symmetric (e.g., CourseEquiv) 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.

Some groups will have to give their UML's in parts. Figure 1 gives the full definitions and context of

- 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.

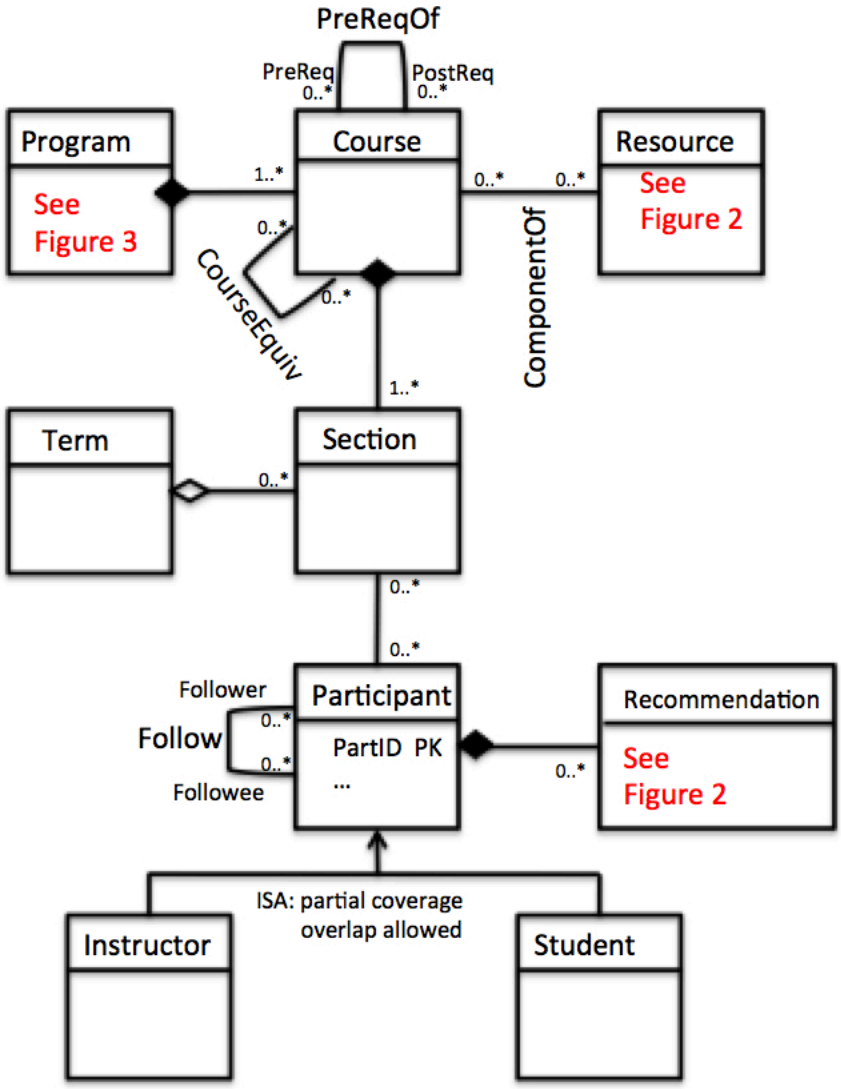

Figure 1: partial UML of student/instructor portfolios

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.

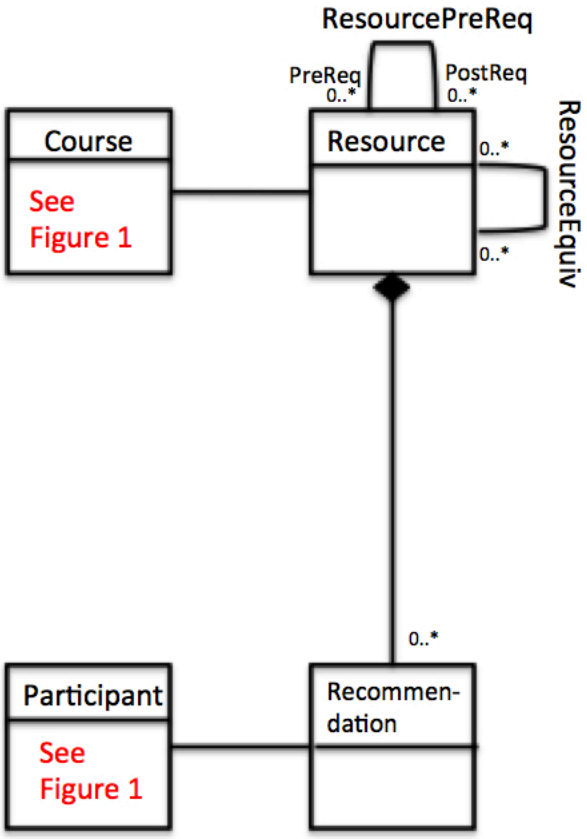

Figure 2: more partial UML of student/instructor portfolios

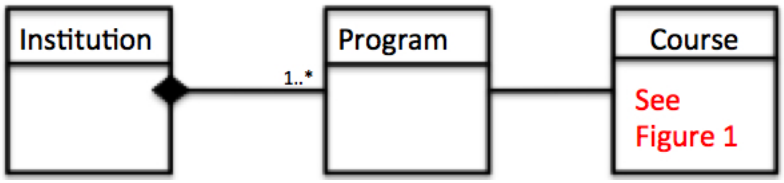

Figure 3: final partial UML of student/instructor portfolios

SECTION 3. Functional Dependencies

List any functional dependencies that do not stem from PK constraints, which you think should be or could be enforced in your database. **Make a note on whether you feel the FD is enforced in your database implementation,** and how if so (as given in the implementation file and at the end of this document). From what I have seen, I do not expect much, if anything, in the way of unenforced FDs. In particular, I think that most of the designs, if not all, represent dependency preserving decompositions of each of your team's "universal relation". But I have also seen cases where a group used auto-increment PKs and I thought that there were possibility other good candidate keys for a table.

SECTION 4. Assertions

List any assertions that you developed as part of the design. You were asked to do at least two in https://my.vanderbilt.edu/cs265/projects/, but many groups did not specify the assertions that I expected them to specify, which were assertions to **enforce 1..* cardinality constraints, disjoint (no overlap), and complete coverage constraints**.

For each of the assertions that you do 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 February 10, which insures that each book (Isbn) participates in association with at least one author through WrittenBy.

```
CREATE ASSERTION BookWrittenByConstraint  /* insure 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)))
```

This policy could be approximated by an in-table CHECK in BOOK, which I show as the very last commented statement in the definition of Book below.

```
CREATE TABLE Book (
  Isbn INTEGER,
  Title  CHAR[120] NOT NULL,
  Synopsis CHAR[500],
  ListPrice CURRENCY NOT NULL,
  AmazonPrice CURRENCY NOT NULL,
  SavingsInPrice CURRENCY NOT NULL,
  ...
  PRIMARY KEY (Isbn),
  FOREIGN KEY (PublisherName) REFERENCES Publisher,
      ON DELETE NO ACTION,  ON UPDATE CASCADE,
  CHECK (Format = 'hard' OR Format = 'audi'
      OR Format = 'cd' OR Format = 'digital')
  /* alternatively, CHECK (Format IN ('hard', 'soft', 'audi', 'cd')) */
  CHECK (AmazonPrice + SavingsInPrice = ListPrice)
/* CHECK (Isbn IN (SELECT Isbn FROM WrittenBy)) */
)
```

In the commented CHECK statement, the outer Isbn refers to the Isbn of a Book record that has just been inserted or updated , and the CHECK is evaluated every time that an insert or update (to Isbn in Book) is made. It only approximates the power of the assertion at top, because this CHECK in Book will NOT be evaluated when a DELETE from WrittenBy is made, which could potentially cause the only association between a Book and an Author to be removed.

Nonetheless, this is an example of how you might mitigate the need for an assertion, or eliminate it entirely with a TRIGGER, in addition to or instead of other within-table constraint mechanisms.

```
CREATE TABLE Author (
  AuthorName     CHAR[120],
  AuthorBirthDate DATE,
  AuthorAddress   ADDRESS,
  AuthorBiography  FILE,
  PRIMARY KEY (AuthorName, AuthorBirthDate)
)
```

```
CREATE TABLE WrittenBy (
  Isbn INTEGER,
  AuthorName CHAR[120],                    /* Consider: how to insure that are no "skipped" */
  AuthorBirthDate DATE,                    /* OrderOfAuthorship beginning at 1 (i.e., */
  OrderOfAuthorship INTEGER NOT NULL, /* disallow entries that represent  first, third, */
  AuthorComment  FILE,                     /* fourth authors (only) for a given Isbn, disallow */
  AuthorCommentDate DATE,                  /* a second author, but no first author, etc. */
  PRIMARY KEY (Isbn, AuthorName, AuthorBirthDate),
  FOREIGN KEY (Isbn) REFERENCES Book,
    ON DELETE CASCADE, ON UPDATE CASCADE,
  FOREIGN KEY (AuthorName, AuthorBirthDate) REFERENCES Author,
    ON DELETE CASCADE, ON UPDATE CASCADE
)
```

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

SECTION 5. 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. However, **REMOVE the names of individuals in your group** from the top of the code listing.

A major component of grading will be to check for consistency between the UML and the table/trigger definitions. Inconsistencies will carry much greater weight than they did in Project 1.

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.

Appendix (of this guide)**: On Justifying Indexes**

In the implementation file (and in section 5 of this document) you should have at least 5 CREATE INDEX statements, and you should free to have more, but you should have header comments for each that index that you have selected, explaining intuitively why is will be beneficial! It is not critical that these be the most important indexes, but the choices should make good sense in terms of the queries that they will speed up (and updates that they might slow down).

I do NOT expect a deep analysis in your header comments for CREATE INDEX statements, but simply evidence that you understand that selection of indexes should be informed by the frequency of queries (and inserts, deletes, and updates) that you expect will be run on the database. If we were just worried about queries (SELECTs) then we might well index everything, but inserts, deletes, and updates can be more costly with stupid indexes, since each indexing structure must also be revised when table entries are revised. Thus we might be more liberal in our use of indexes in a table where inserts and deletes are relatively rare (e.g., the Books table in our illustrative database), than in the Transactions table (inserts frequent).

Professor Widom spoke of sophisticated software that could select indexes automatically given a set of queries, inserts, deletes, updates (call this set O for "operations"). Roughly speaking, this software will select an index if the expected cost associated with using the index is less than the expected cost of NOT using it, or:

$$\text{ExpectedCostSavings(Index I)} = \Sigma_O \; P(O)[\text{Cost}(O,\sim I) - \text{Cost}(O,I)],$$

where P(O) is the estimated proportion of time O is executed over all operations in the workload; Cost(O,~I) is an estimate of the cost of executing O without the index, I; and Cost(O,I) is an estimate of O's cost with I. You can imagine that to be more accurate, this software would consider the effect of multiple indexes simultaneously, rather than considering them independently as above, so if you've had the AI class before, you can probably see the relevance to some of the methods studied there – search, constraints, optimization, planning.

But, I digress. Suffice it to say that **when you choose an index, justify it by explaining how it will increase the efficiency of frequent queries**. In doing so, you can point to queries that you have actually given in your implementation file (and section 5 of this document). Also, as part of the justification, mention inserts, deletes, and updates that will be slowed by the index (think B+ tree node splitting and extendible hash table bin splitting); the operations that are slowed may be relatively infrequent compared to the queries that are speeded up, but it may be more that the queries are speeded up by huge amounts on average and "updates" (inserts, deletes, updates) are slowed only a wee bit.