Healthcare Database Management for Health Informatics and Information Management Students: Challenges and Instruction Strategies—Part 2

by Ray Hylock, PhD, and Susie T. Harris, PhD, MBA, RHIA, CCS

Abstract

In the first part of this two-part series, the methodology behind the creation of a graduate-level course in healthcare database systems at East Carolina University was presented. The distinction between healthcare and traditional database courses was examined, along with construction of content. Specifically, minimum and extended construct sets pertaining to database modeling and interrogation were defined, taking into consideration accreditation requirements and a desired level of skill. The focus of this article is on implementation details, including course responsiveness and problem deconstruction strategies. Suggestions and examples for challenging advanced students without disturbing the overall course grade distribution are submitted. Lastly, 10 synthetic healthcare data sets are provided. These sets were generated on the basis of real-world patterns seen in healthcare data, vary in size from 10 patients to 1 million patients, and are available in general as well as database management system–specific formats for those recommended in the first article in the series. The result of this work is an implementation framework, complete with examples and sample data sets, for the dynamic delivery of a healthcare database course intended for a health informatics and health information management audience.

K**eywords:** healthcare databases; healthcare database instruction; health information management (HIM); health informatics (HI); health information technology; database course development/design; HI/HIM instructional design; HI/HIM course development; health information technology instructional design

Introduction

The first installment[1] of this series addressed the necessity of database instruction for health informatics (HI) and health information management (HIM) students because of the pervasiveness of healthcare information systems,[2–4] critical work area foci defined by the American Health Information Management Association (AHIMA) HIM Workforce Study,[5] and Commission on Accreditation for Health Informatics and Information Management Education (CAHIIM) accreditation standards.[6] In that article, a lecture outline mapped to CAHIIM baccalaureate and master's degree competencies was proposed, with justification for each topic. Additionally, minimum and extended construct sets were posited for database modeling and interrogation. Lastly, database management system features for education were submitted, complete with evaluations of commonly used tools and recommendations for student-managed and university- or instructor-hosted environments.

In this work, the second of two parts, the focus is on implementation strategies. Of greatest import is defining the depth and breadth of content, requiring the instructor to understand the backgrounds and abilities of the students. For this purpose, the authors have developed a comparison matrix to categorize students based on the extent of their technical backgrounds and abilities. The aggregated scores establish the level of course content, ensuring that all minimum standards are met. As the semester progresses, content may be updated to reflect student performance. To facilitate this pedagogical design, the nature of the course must be responsive to changes at any given point. Likewise, the material must challenge students with prior experience in the area, while not disrupting the overall grade distribution of the course (i.e., by favoring prior knowledge) or going topically beyond the capabilities of the students in general. Techniques for achieving these goals are provided in this article.

Once a responsive course of appropriate depth and breadth is established, the subject matter must be presented and evaluated. This work proposes two instructor-directed approaches—*piecewise incremental* and *piecewise combination*—to combat the pitfalls of using the commonly accepted *all-at-once* and *incremental* dissemination and evaluation methods in the teaching of database modeling and interrogation. The goal is to maximize student comprehension, retention, and point totals, while minimizing confusion, angst, and error propagation. In line with the minimalist theory[7] discussed in Part 1 of this series,[8] error recognition and recovery examples are also provided to enhance the learning experience. Limiting the discussion to database modeling and interrogation (as in Part 1[9]) is justified because these two topics are the most commonly reported database curriculum elements of particular difficulty.[10–13] The rationale for their difficulty is intuitive. Database modeling involves envisioning an interconnecting set of logically related data collections, whereas database interrogation (typically achieved by way of Structured Query Language [SQL]) requires the ability to think procedurally. These are not skills one simply happens upon in management or professional degree programs. This reality is exacerbated by the considerable variation in student backgrounds and abilities in HI and HIM programs.

The last contribution of this article pertains to data sets for database instruction. Acquiring healthcare data is generally fraught with challenges, and generating internally consistent information can be overwhelming. As a result, many instructors rely on data sets containing a relatively small number of tuples (records) or utilize nonhealthcare data sets that accompany textbooks. Studies routinely show the importance of using semirealistic data sets to effectively teach the use of databases.[14, 15] This means that the data should be domain specific and of reasonable scale. Thus, 10 synthetic data sets of varying size specifically designed for healthcare database instruction, generated according to real-world healthcare data patterns, are provided.

Student Backgrounds and Abilities

HI and HIM careers span a wide swath of professions; hence, student backgrounds and abilities vary. The assorted programs educate those with and without healthcare experience—in fields ranging from accounting and sociology to nursing and medicine—and with varying technical expertise, from those without a technical background to database administrators. These technical and experiential knowledge gaps are quite common in HI and HIM programs, and they present two key challenges in the more technical courses, such as database management systems.

The first challenge concerns course responsiveness. Disciplines such as computer science and information systems have strict guidelines for program admittance, including prerequisite knowledge in technical areas such as programming and modeling.[16, 17] Thus, database courses can progress each semester at a similar pace because the cohorts are homogeneous. As stated, this does not hold true in our discipline, necessitating a responsive course design. The second challenge pertains to maintaining a graduate-level pace without overwhelming students lacking technical knowledge, while stimulating those with such skills. These challenges will be addressed in detail throughout the remainder of this article.

Each semester, students who register for the database course complete an introductory and background survey before the start of the term. Questions pertain to prior degrees, current program, work and academic database experience, computer science background, and perceived technical prowess. This information allows the instructor to discern two key attributes: theoretical background and technical ability. The comparison matrix in Table 1 provides further clarification of the student classification based on these dimensions.

Course Responsiveness

The structure of most courses is rigid, with predefined modules, learning outcomes, and lecture content. General responsiveness (i.e., course flexibility) might encompass assignment extensions or a few minutes spent reiterating materials from previous lectures. In most cases, this flexibility is acceptable and sufficient. In HI and HIM database courses, however, the abilities and backgrounds of students leads to greater uncertainty, necessitating a truly responsive design. Similar in premise to a computerized adaptive test, a course with responsive design adapts the difficulty of course content, concept depth, and assignments on the basis of students' cumulative progress.

Beginning with the minimum construct set defined in Part 1,[18] extended construct set material can be injected depending on students' overall perceived and achieved performance. For instance, if students have successfully implemented strong and weak entities (a concept in the minimum construct set) with minimal difficulty, supertypes and subtypes (a concept in the extended construct set) can be introduced. This is done by preparing material for all concepts in the extended construct set (e.g., lecture slides, examples, assessment questions, etc.) and identifying queues for subject inclusion. Essential to the success of this form of design is the student survey.

The base instructional level is defined by the minimum construct set. Student survey results are then applied to adjust the initial level of the course. As the term proceeds, student classification and overall progress dictate future modifications, never straying below the defined minimum construct set.

Planning Course Modifications

The adaptive materials are to be defined and created during course preparation, as it is imperative to fully understand the implications of adding a topic and its effect on course progress. The following example illustrates this idea.

Student surveys indicate very little in the way of theoretical background, yet the students seem to grasp the concept of indices with relative ease. As a result, the instructor decides to introduce index data structure declarations. Students are provided with a basic introduction and Data Manipulation Language (DML) structure, and then they solve several exercises. The problem is that students with limited theoretical background in computer science or information science generally fail to fully comprehend the impact that the selected data structure has on system performance. It is not uncommon for students to misconstrue the nature of a bitmap index as one of general optimization rather than for its intended purpose of enhancing query performance in primarily analytic environments. As a result, students try to incorporate bitmaps into their database design, which generally hinders performance in high-volume transactional environments (e.g., hospitals and clinics). When inserting, updating, and deleting records becomes cumbersome, students become confused, point totals decrease, and lecture time is lost trying to explain a construct that is of little value in an introductory database course. This example demonstrates that a thorough examination of each topic in the extended construct set for its impact on existing content, prerequisite knowledge (both background knowledge via overall classification of students taking the course [see Table 1] and knowledge of the presented subject matter), objectives, and measures is required.

This process begins with course objectives, an example of which is *construct conceptual data models*. Course objectives are designed to meet accreditation requirements, such as those defined by CAHIIM.[19] From course objectives, learning objectives are identified. Continuing with the example, a learning objective for *construct conceptual data models* is *strong and weak entities*. Lastly, an appropriate measure(s) for the learning objective must be determined. In this case, the measure is simply the *appropriate use of entity types*. This is the most basic unit of evaluation and will assist in determining the success or failure of a learning objective.

Tables 2, Table 3, Table 4, and Table 5 are a detailed breakdown of course objectives, learning objectives, and measures for the minimum and extended construct sets that were defined in the first article in this series.[20] Associated with each extended construct set measure is the minimum background and ability quadrant, as expressed in Table 1. If a thorough analysis had been performed, taking into consideration the minimum background/ability classification for index data structures outlined in Table 3 (i.e., TT), the situation presented in the example would have been avoided.

Engaging the Advanced Student

The need to keep advanced students engaged is a delicate and sometimes overlooked subject. In classes with such a diversity of backgrounds, it is important to engage advanced students without jeopardizing the learning potential of the rest of the class. The goal is to incentivize topic exploration without skewing grade distributions. In the following "Problem Deconstruction" and "Error Recognition and Recovery" sections of this article, detailed instances of this topic will be presented. However, to highlight some of the techniques, a few examples are as follows:

- *Unary joins*—The concept of a unary join is somewhat difficult for students to comprehend. The trouble lies in the notion of an object being related to itself. For students with a more technical background, especially in programming, the idea is easily related to recursion. To those without a technical background, however, the time required to explain the matter would be better used to reinforce basic concepts. Thus, for a few extra points, one can entice the advanced student to attempt the problem.

- *Function construction*—Aggregate functions, such as average and count, are performed by virtually every database management system. Expanded function sets like those containing statistical tools (e.g., standard deviation), however, are not. Therefore, constructing functions by parts is an integral skill for advanced database users. To promote such a process, one could give students the option of computing an aggregate function, such as average, by its constituent parts—sum and count.

- *Views*—Views are an efficient and effective way to control data access, minimize the complexity of SQL queries, and mask sensitive bits of information. To encourage the study of views, one could provide a few extra credit points to first define a view for a specified subpart and then integrate that view in a final SQL statement, as opposed to a single solution.

Problem Deconstruction

Reviewing the database modeling and SQL exercise sections of most textbooks reveals a consistent theme—single statement construction.[21–27] This follows the general pattern in textbooks of defining a single problem instance, then having the student either solve the problem *all at once* or perform a logical *incremental* exercise. For students lacking familiarity with programming, solving simple problems can be demanding, let alone being required to parse ones that are complex. Even advanced students have difficulty deconstructing problems into pieces without experience. Therefore, students often fail to complete assignments and reinforce the material, not because the students cannot generate simple commands, but because they are unable to dissect intricate problem statements. Consequently, grades often reflect the ability of a student to analyze problem structure instead of preparing solutions—effectively penalizing those new to the topic. The question becomes, how can we, as database instructors, not only assist the students in solving the questions asked, but also determine the exact pieces causing the most difficulty? The answer is the *piece-wise deconstruction* approach.

A simple yet often difficult concept to master, piecewise deconstruction is the systematic construction of problem instances that are easily deconstructed into objective-based sub-elements that are more readily solvable. Each sub-element constitutes one or more measures for the learning objective assigned to the problem. The challenge for the instructor is to create self-contained or minimally incremental subproblems that fit together in a coherent and easily followed manner. As a result, this process is sometimes iterative.

Table 6 describes four approaches for presenting problems to students. The first two (*all at once* and *incremental*) are widely found in database textbooks and subject to the issues previously discussed. The final two (*piecewise incremental* and *piecewise combination*) are approaches implemented in the database course.

Piecewise combination requires independent problems to be solved individually and then combined to produce the intended results. Points are weighted heavily in favor of the independent pieces, with minimal points awarded for combination (which generally requires simple statement integration over assembly). The following three-part example illustrates this concept (parenthetical statements are the measurable objectives).

(1) Return male (M) patients – 2 points (single-table select and unary filter).

```
SELECT * FROM patients WHERE gender = 'M';
```

(2) Return patients born in the year 2000 – 2 points (single-table select and range filter).

```
SELECT * FROM patients
WHERE birth_date BETWEEN '2000-01-01' AND '2000-12-31';
```

(3) Return male patients born in the year 2000 – 1 point (combine (1) and (2)).

```
SELECT * FROM patients WHERE gender = 'M'
AND birth_date BETWEEN '2000-01-01' AND '2000-12-31';
```

The piecewise incremental approach is a form of problem construction in which one piece builds on its predecessors. In contrast to the *incremental* approach, the piecewise incremental approach presents students with the incremental steps as independent problems to be solved. Thus, the instructor can account for error propagation, which, in this context, is defined as the effect that previous errors have on the current solution. Let us use the following two-part piecewise incremental example to explain this process. As in the previous example, parenthetical statements are the measurable objectives, with one caveat: only the incremental change is measured (e.g., problem 2 simply adds a range filter to the single-table select and unary filter measures of problem 1).

(1) Return male (M) patients – 2 points (single-table select, unary filter).

```
SELECT * FROM patients WHERE gender = 'M';
```

(2) Beginning with (1), return patients born in the year 2000 – 2 points (range filter).

```
SELECT * FROM patients WHERE gender = 'M'
AND birth_date BETWEEN '2000-01-01' AND '2000-12-31';
```

In the remainder of this section, database modeling and SQL examples for these approaches are presented.

Database Modeling

In database modeling, students must analyze a process, determine data collection requirements, normalize relations resulting from the prior phase, and provide the appropriate level of associations in the diagram. Many have never experienced such an abstract construct, and therefore have difficulty perceiving the information in that context. Thus, piecewise deconstruction promotes not only greater solvability of a problem, but student comprehension as well. In this section, a piecewise combination example is provided for a simple healthcare scenario, accompanied by suggestions for engaging advanced students.

Piecewise Approach

Example—Generate a conceptual model to answer the following question (piecewise combination): Alliance Regional Medical Center is developing a simple disease and problem list registry for analytic purposes. The registry will track patients through encounters, recording their disease and problem list entries, if any. Diseases and problems are to be associated with the assigning encounter and a lookup table of ICD-10-CM codes. (A description of collected attributes is generally given here but is withheld for the sake of brevity.)

Although the problem appears simple, students quickly become overwhelmed trying to conceive of entity interrelatedness without segmenting the statement (i.e., an all-at-once approach). Typically, the `Patients` and `Encounters` entities are correctly created; it is the difference between `Diseases` and `Problem_List`, and the role of `ICD10CM`, that cause confusion. This is not an artifact of the assignment, but an intentional infusion of beginner and intermediate concepts to challenge the students and expose them to shared entities and relationship semantics.

The following example illustrates a piecewise combination approach to solving this problem. Of note, shared entities and relationship semantics become apparent during the final combination event. That is, `Encounters` and `ICD10CM` share two paths: one through `Diseases` and the other through `Problem_List`. The latter chronicles long-term diseases (e.g., diabetes or heart disease), while the former records all others.

1. *Provide the conceptual model for Patients and Encounters – 3 points (one-to-many and mandatory patient inclusion).* See Figure 1(a).
2. *Provide the conceptual model for Encounters and Diseases – 5 points (many-to-many and optional disease inclusion).* See Figure 1(b).
3. *Provide the conceptual model for Encounters and Problem List – 5 points (many-to-many and optional problem inclusion).* See Figure 1(c).
4. *Provide the conceptual model for Patients, Encounters, and Diseases – 1 point (combine (1) and (2)).* See Figure 1(d).
5. *Provide the conceptual model for Patients, Encounters, Diseases, and Problem List. Hint: five entities and five relationships – 2 points (combine (3) and (4)).* See Figure 1(e).

Engaging the Advanced Student

The conceptual model presents many opportunities to challenge a student, two of which are as follows. First, one can ask the student to add a relationship to reduce query processing when retrieving problems for patients. The solution is to add a relationship between `Patients` and `Problem_List`, bypassing `Encounters`. Second, one can require the capture of the primary disease for each encounter. This necessitates a one-to-one relationship between `Encounters` and `ICD10CM` and has proven challenging for students—its difficulty lies in its simplicity.

Structured Query Language

Database interrogation (typically achieved by way of SQL) requires the ability to think procedurally. SQL is classified as a fourth-generation programming language, meaning that it is more English-like in syntax than others, but learning it still presents many of the same challenges as learning other languages. For those inexperienced in programming, a simple `SELECT` statement might be overwhelming, let alone a multiline question. This section provides several examples of piecewise problem deconstruction to assist students in both understanding and solving SQL queries, along with suggestions for engaging advanced students.

Piecewise Approach

Example 1—Return all patient and encounter details for the patient(s) with the youngest recorded encounter age (piecewise incremental).

While appearing unsophisticated, the solution requires two-deep nesting. Most students new to SQL try to write from the outside in, starting with what must be returned. For instance:

```
SELECT * FROM patients p, encounters e
WHERE p.patient_id = e.patient_id;
```

This rarely provides trouble for the student (if it does, the error is usually a forgotten join predicate). The next step is to find the youngest patient. Again, this appears straightforward, but it is actually nontrivial. The student must initially determine the youngest encounter age:

```
SELECT MIN(age) FROM encounters;
```

Then the student has to integrate this information into the previous template. Here is where the struggle usually lies. The correct approach is to use a subquery:

```
SELECT * FROM patients p, encounters e
WHERE p.patient_id = e.patient_id
AND e.age = (SELECT MIN(age) FROM encounter);
```

Is this the final solution? Most students think it is. The youngest patient(s) encountered has been identified and the appropriate details returned. The correct answer, however, is no. This solution only provides the information for the encounter in which the patient(s) had the minimum age, not *all* encounters for the patient(s). For the instructor, the question when grading is “Did the student not understand the question, or did the student fail to answer it properly?” Answering this question is challenging, to say the least. Hence, as with conceptual modeling, SQL queries should be provided in piecewise form.

The previous question can be modified by asking the student to solve it in three piecewise incremental parts: (1) find the youngest encounter age, (2) find the patient(s) with that encounter age, and (3) find all encounters for the specified patient(s). These parts build on the previous solution in reverse nested order (i.e., inner to outer), which is the correct approach when solving nested queries. However, students new to databases rarely think from the inside out because it is not natural and requires forethought about how to solve such questions. Thus, this ordering is also designed as an instructional aid, illustrating the thought process for solving such problems. This takes planning on part of the instructor, but appreciable gains in points and understanding from the students will be seen. The rephrased question is as follows:

Return all patient and encounter details for the patient(s) with the youngest recorded encounter age. Solve in parts.

1. Return the youngest age in encounters.
2. Beginning with (1), return the patient ID(s) for the encounter with the youngest age.
3. Beginning with (2), return all patient and encounter details for the patient(s) with the youngest encounter age.

The parts support the two requirements. First, they simplify the query into pieces more straightforward to solve. Second, errors are more readily identifiable, with two positive outcomes: first, accounting for error propagation allows for greater partial credit to be awarded, and, second, concrete instances of errors become available for class discussion and course modification.

Here is the solution:

(1) Return the youngest age in encounters – 2 points (single-table select and full set aggregate).

```
SELECT MIN(age) FROM encounters;
```

(2) Beginning with (1), return the patient ID(s) for the encounter with the youngest age – 3 points (single-table select, unary set filter, and noncorrelated subquery).

```
SELECT patient_id FROM encounters
WHERE age = (SELECT MIN(age) FROM encounters);
```

(3) Beginning with (2), return all patient and encounter details for the patient(s) with the youngest encounter age – 5 points (inner join and inclusion set filter).

```
SELECT * FROM patients p, encounters e
WHERE p.patient_id = e.patient_id
AND e.patient_id IN (SELECT patient_id FROM encounters
  WHERE age = (SELECT MIN(age) FROM encounters));
```

There are still challenges (e.g., the correct use of `MIN` in (1), the subquery in (2), and the `IN` clause in (3)), but the problem is more straightforward, allowing the student to focus on the queries instead of parsing the problem into solvable elements.

Another useful tool is providing hints for queries. For instance, the instructor could give a range for the result in (1) (e.g., the youngest age is between 0 and 1), the number of results in (2) (e.g., two results), or a detail from the result output in (3) (e.g., one of the encounters will have an admit date of 2017-01-12). These hints do not find the solution for the student, but they provide security to the student when an answer is finally produced within the bounds of the hint.

Example 2—De-identify patients following the Safe Harbor method defined in 45 CFR §164.514(b)(2) of the Privacy Rule (piecewise combination with piecewise incremental substeps).

Students must first produce a de-identified map for `Patients`. Because this section emphasizes SQL challenges, the solution to the conceptual modeling phase is provided in Figure 2 as a foundation for this exercise. The map clearly shows the discarding of five attributes, retention of one attribute, and transformation of three attributes. The latter necessitates a surrogate key substitution for `patient_id`, binned year from `birth_date`, and masked `zip`. The solution is as follows.

(1) Create the `target` schema – 1 point (create schema).

```
CREATE SCHEMA target;
```

(2) Create the target table – 3 points (create table, constraints, implicit primary key index).

```
CREATE TABLE target.patients (
  patient_id     INTEGER NOT NULL,
  birth_year     VARCHAR(7),      -- max of 7 characters: '<= 1927'
  state          CHAR(2),         -- two-character abbreviation
  zip            CHAR(5),         -- masked zip
  CONSTRAINT patients_pk PRIMARY KEY (patient_id)
);
```

(3) Provide the identified base template using `source` – 1 point (single-table select).

```
SELECT patient_id, birth_date, state, zip FROM source.patients;
```

(4) Generate de-identified key map – three steps.
(4i) Create a table to generate and supply patient surrogate IDs – 2 points (create table).

```
CREATE TABLE source.patient_ids (
  patient_id    INTEGER, -- original id
  patient_id_d  SERIAL   -- generated, de-identified id
);
```

(4ii) Populate the map – 3 points (insert via single-table select).

```
INSERT INTO source.patient_ids (patient_id)
SELECT patient_id FROM source.patients;
```

(4iii) Select the new patient IDs from the map – 2 points (inner join).

```
SELECT patient_id_d FROM source.patients p, source.patient_ids i
WHERE p.patient_id = i.patient_id;
```

(5) Extract year from `birth_date` and bin – 4 points (`CASE`, `CAST`, and PostgreSQL's `EXTRACT` functions; single-table select; unary operator; and constant values).

```
SELECT CASE WHEN EXTRACT(YEAR FROM birth_date) <= 1927 THEN '<= 1927'
ELSE CAST(EXTRACT(YEAR FROM birth_date) AS VARCHAR(7)) END AS
birth_year
FROM source.patients;
```

(6) De-identify zip codes.
(6i) Compute the population `SUM` for the first three digits of the zip code – `source.zips` provides zip code–population pairings from the US Census Bureau – 3 points (unary division and subset aggregate).

```
SELECT zip/100, SUM(population) FROM source.zips GROUP BY zip/100;
```

(6ii) Add the 'xx' mask option – the displayable de-identified solution – 1 point (concatenate with constant value).

```
SELECT zip/100, zip/100 || 'xx', SUM(population)
FROM source.zips GROUP BY zip/100;
```

(6iii) Add the '000xx' option – the nondisplayable de-identified solution – 1 point (constant value).

```
SELECT zip/100, zip/100 || 'xx', '000xx', SUM(population)
FROM source.zips GROUP BY zip/100;
```

(6iv) Select the appropriate mask (based on the 20,000 threshold) for each zip using the results from (6iii). This produces a Safe Harbor aligned lookup table – 3 points (`CASE` function and unary operator).

```
SELECT zip/100, CASE WHEN SUM(population) > 20000 THEN zip/100 || 'xx'
ELSE        '000xx' END AS mask
FROM source.zips GROUP BY zip/100;
```

(6v) De-identify zip using the lookup table in (6iv) – 3 points (inline-select and inner join).

```
SELECT z.mask FROM source.patients p, (SELECT zip/100 AS zip, CASE
WHEN        SUM(population) > 20000 THEN zip/100 || 'xx' ELSE '000xx'
END AS mask
       FROM source.zips GROUP BY zip/100) AS z
WHERE p.zip/100 = z.zip;
```

(7) Replace identified attributes in (3) with their de-identified counterparts in (4)–(6) – 2 points (combine) – to conserve space, only the substep definitions are presented: (7i) combine (4iii) with (3), (7ii) combine (5) with (7i), and (7iii) combine (6v) with (7ii).

(8) Load the data from (7iii) into the table created in (2) – 1 point (insert via multitable select).

```
INSERT INTO target.patients (query from (7iii));
```

This singular problem not only allows the instructor to measure 16 distinct objectives and is easily adjusted for error propagation, but is presented to students in simple fragments that allow them to focus on solving the problem in easily comprehendible pieces. From experience, students are far less frustrated, are generally able to solve the problem in its entirety, and have more specific questions for the instructor. The latter is quite important given the prevalence of distance education in HI and HIM programs.

Engaging the Advanced Student

Two options to engage the advanced student are evident. First, one can provide a few extra credit points for the creation of a view based on (6iv) used in (6v), 7(iii), and (8). The solution with and without the view is the same; however, the view requires an additional Data Definition Language (DDL) statement and three DML modifications. Second, one can ask the student to provide a one-line addendum to (8) to guarantee the use of a primary index on `target.patients.patient_id`. The student must first understand the requirements of a primary index (i.e., unique data stored sorted) to augment the statement. Because (8) loads *all* data, only the order must be explicitly controlled.

Error Recognition and Recovery

Error recognition and recovery is a key principle in the minimalist theory.[28] Through the analysis of incorrect solutions, students challenge and reinforce their understanding of the materials. This section focuses on techniques for logical error recognition and recovery. Syntactic errors, while important, are beyond the scope of this article, because correction of those errors is a simple debugging exercise, rather than requiring the student to locate bits of models and statements that are syntactically correct yet produce logically fallible results. Following the focus of this article, database modeling and SQL examples are presented.

Database Modeling

This section offers two examples of the error recognition and recovery process for conceptual modeling.

Example 1—Inpatient/outpatient procedures example: Figure 3 illustrates a simplified process for recording procedures during encounters. Determine if there are any scenarios possible that cannot be captured by the current configuration. If so, describe the recovery.

Left to their own devices, students largely fail to see the logical error as they are more focused on the correctness of the diagram (i.e., syntax). To assist the students in this process, the instructor can ask, “How many procedures can one have during an encounter?” The answer, of course, is many. Following up, the instructor can ask, “How many *identical* procedures can one have during an encounter?” This usually draws quizzical looks because students might not understand why one would have the same procedure multiple times during an encounter. Why is this? Students typically think of encounters through an outpatient rather than inpatient lens. That is, the procedure is an encounter in and of itself, rather than an encounter being a block of time, as can happen in an inpatient setting. An illustration of this is the treatment of a burn victim. During the patient’s stay in the hospital (i.e., encounter), the patient may undergo multiple skin grafts. Therefore, the model in Figure 3 must support repeat occurrences of a procedure. Now, the students are primed for error recognition. Returning to the figure, it is clear that identical procedures during an encounter are *not* supported. The recovery step is to move the date (which includes time by definition) to the primary key.

Example 2—Inpatient/outpatient procedures example with inventory: Figure 4 illustrates a simplified process for recording inventory utilization during a procedure. Determine if there are any scenarios possible that cannot be captured by the current configuration. If so, describe the recovery.

As with the previous example, students will likely note the correctness of the model, in that each procedure has the ability to record an inventory item. The question then becomes "How many times can an item be used during a procedure?" According to the model, once. Is this enough? Does this mean only one stitch or piece of gauze can be used? Unfortunately, that is a limitation imposed by the presented model (error recognition). To recover from the error, each inventory item used during a procedure should include a simple quantity attribute.

Structured Query Language

The following section provides examples of SQL error recognition and recovery. Statements are provided along with a discussion of common student mistakes, leading questions for the instructor to ask, and opportunities for topic exploration.

Example 1—Boolean parenthetical analysis: The following statement has been provided in response to a request for all encounters seen in any clinic or radiology on or after January 1, 2017.

```
SELECT * FROM encounters
WHERE dept_name LIKE 'clinic%'
OR dept_name = 'radiology'
AND admit_date >= '2017-01-01';
```

Students typically read through the statement and declare it correct. It is only after execution and examination of the `Encounters` table that the problem becomes apparent. The questions for the instructor to ask are "Is it correct and if not, why?" (error recognition) and "How can it be corrected if necessary?" (recovery). This is an excellent opportunity to introduce the `EXPLAIN` statement. The query produces the following (simplified) execution plan: ('clinic%') OR ('radiology' AND >= '2017-01-01'). With this, the error becomes evident. The intent is to filter departments and *then* dates. Predicates, however, are segmented by OR instead of AND by default; hence the logically incorrect results. Moving to the second question, recovery is achieved by enclosing the department predicates in parentheses, resulting in the intended execution plan of ('clinic%' OR 'radiology') AND (>= '2017-01-01').

Example 2—Foreign keys: Beginning with the following table creation command, (1) identify and justify current constraints and (2) propose, if necessary, additional constraints to control for data integrity and validity.

```
CREATE TABLE encounters (
  encounter_id   SERIAL, -- PostgreSQL auto-increment data type
  patient_id     INTEGER NOT NULL,
  CONSTRAINT encounters_pk PRIMARY KEY (encounter_id)
);
```

This example provides an opportunity for cascade error recognition and recovery. That is, recovering from an error presents a subsequent one. For even the novice student, the first question is easily solved—there exists a primary key constraint controlling for entity integrity. The second, however, is not easily solved in its entirety. The explanation is as follows.

Students can usually deduce the logical error with the help of leading questions such as "For whom is the encounter?" or "Who is billed?" They realize that `patient_id` is not declared as a foreign key (error recognition) and quickly produce the corrected statement (recovery). The question then becomes "Is this the only error?" Most students say yes. That is not necessarily correct. To be certain, one must successfully implement the foreign key, which ensures that all referenced values are valid. From an instructional standpoint, this provides an excellent opportunity to present the student with both valid and invalid foreign key data. The setup is quite simple: create a table utilizing only existing patient IDs and another that does not. The first will process the foreign key constraint without issue (full recovery), whereas the second will fail (recovery leading to another error). To assist with error recognition, one can extract encounters assigned to nonexisting patients (see the query below—while the least optimal of the `ANTI SEMI-JOIN` varieties, it is easier for novices to comprehend than the correlated `NOT EXISTS` and precarious `NOT IN` variants). From there, the student must take the appropriate corrective action for the invalid patient IDs in `Encounters` *before* remedying the original logical error.

```
SELECT * FROM encounters e LEFT OUTER JOIN patients p USING
(patient_id)
WHERE p.patient_id IS NULL;
```

Healthcare Instructional Data Sets

The acquisition of data sets for instruction is a difficult task regardless of the field of study. Textbooks often provide easy-to-understand database examples that, unfortunately, are highly simplified—referred to as "toy" databases. The initial introduction to a topic warrants a brief example; however, these database examples do not reflect reality. According to Yue,[29] the average number of relations in a data set example from an information system–focused database textbook is 7.2, with 14.3 tuples per relation. Yue's data were drawn from Teradata University[30]—a commonly used data set distribution resource for database textbook publishers. The authors compiled data from six textbooks over 13 of the available 16 data sets. They chose the two larger sets rather than the two smaller sets from Jukić, Vrbsky, and Nestorov[31] but the smaller of the two sets from Hoffer, Ramesh, and Topi.[32] Expanding the analysis to include 13 working textbooks (Watson's *Data Management: Databases and Organizations*, fifth edition,[33] does not function) and 29 data sets, the average number of relations increases to 11.8 (median, 8; standard deviation, 7.3), and the average number of tuples increases to 96.12 (median, 9; standard deviation, 479.15) respectively. While 8 to 12 relations are more than adequate, assuming sufficient schema complexity, having only 9 to 96 tuples is unacceptable.

Studies have routinely stressed the importance of teaching databases in semirealistic, scaled environments to expose the student to challenges faced in professional settings, such as SQL profiling and optimization, performance concerns (e.g., memory, multilevel indices, and join/predicate ordering), and de-normalization for analytic tasks.[34, 35] Therefore, it is paramount to provide a semirealistic, scaled environment to prepare students for real-world database interaction. Here is where the problem lies. Where can one get healthcare data, let alone at sufficient scale?

To the best of our knowledge, a healthcare-specific database textbook does not exist. Thus, instructors are forced to either look to industry for data sets or create their own. The former can be challenging because healthcare data are highly protected by the collecting institution. Many institutions are unwilling to share data (even de-identified) because of fear of litigation and/or HIPAA Privacy Rule and/or Security Rule violations—especially when the data are to be disseminated to parties outside of their control (e.g., students). If data are provided, the users generally sign a data use agreement and might not be allowed to load the data on a machine outside of a controlled network—an untenable position for distance education instruction.

Creating one's own data set for instruction is a viable alternative, but unlike simple business data, healthcare data are highly complex and interdependent. For instance, demographics, vital signs, and current and past diagnoses and medications must be taken into consideration when generating the next record (e.g., medication). The best course of action is to model relationships and patterns found in existing data. Of course, one must have access to such data first. Fortunately, the authors have had access to various real data sets over the years from which said relationships and patterns were extracted. Using this information, 10 synthetic data sets have been produced and are available for download. The schema is shared across all instances and contains eight tables. The sets are defined by the number of patients, which range from 10 for simple examples to 1 million for scaling and optimization. The sets can be downloaded in PostgreSQL backup files, MySQL data exports, or plain SQL. For more details, see Appendix A.

Conclusions

Student diversity in HI and HIM programs necessitates the appropriate design and configuration of content in technical courses. Each semester brings a new cohort of students with heterogeneous backgrounds and abilities in computer-based and computational fields such as databases. In this article, the second of a two-part series, three implementation strategies are introduced.

The first involves the establishment of an instructional baseline for the course. A comparison matrix of students' varying technical or nontechnical backgrounds and abilities is used to evaluate individuals, culminating in an overall assessment of the class. While never straying below a minimum threshold of instruction, the instructor can use the baseline assessment to forecast the depth and breadth of materials at the start of the semester. As shown, this baseline is adjusted to reflect student performance throughout the term.

Adjustments to the course content are achieved by way of the second strategy, which involves designing a truly responsive course. This process begins with extensive planning and analysis of each topic to be covered. Building on the minimum and extended construct sets introduced in Part 1,[36] each course objective is deconstructed into learning objectives and measurable outcomes. The learning objectives are associated with construct sets, and the measurable outcomes are associated with quadrants of the student comparison matrix, effectively mapping concepts to students' backgrounds and abilities. This results in a detailed list of topics that can be injected into the course depending on the instructor's perception of the students' level of comprehension. Furthermore, it provides a means to challenge advanced students.

The third strategy is problem deconstruction by means of the proposed piecewise techniques. The prevailing all-at-once and incremental approaches fail to accommodate students' struggles while learning a topic. When the all-at-once approach is used, a student can become overwhelmed, and a single early mistake can completely sabotage the outcome. The incremental approach requires the student to break down a problem into subparts, which can result in a cascading failure. Both of those approaches prevent the instructor from truly measuring student understanding because each error has several logical precursors. Thus, the proposed piecewise approaches provide a distinct advantage over extant methods by

1. providing students with more readily solvable subquestions,
2. allowing for direct measurement of associated outcomes,
3. accounting for error propagation by enabling the instructor to locate and track errors, and
4. providing evidentiary support for future modifications to the course and the manner in which it is taught.

The end result of employing the three implementation strategies and data sets discussed in this article, along with the instructional design conveyed in Part 1, is a course that maintains a minimum level of education, challenges advanced students, and maximizes the learning potential of each unique cohort of students.

Ray Hylock, PhD, is an assistant professor in the College of Allied Health Sciences at East Carolina University in Greenville, NC.

Susie T. Harris, PhD, MBA, RHIA, CCS, is an associate professor and director of the Health Informatics and Information Management Program at East Carolina University in Greenville, NC.

Notes

1. Hylock, Ray, and Susie T. Harris. “Healthcare Database Management for Health Informatics and Information Management Students: Challenges and Instruction Strategies—Part 1.” *Educational Perspectives in Health Informatics and Information Management* (Winter 2016): 1–24.
2. Jacob, Julie A. “HIM’s Evolving Workforce: Preparing for the Electronic Age’s HIM Profession Shake-Up.” *Journal of AHIMA* 84, no. 8 (August 2013): 18–22.
3. American Health Information Management Association. “Embracing the Future: New Times, New Opportunities for Health Information Managers. Summary Findings from the HIM Workforce Study.” June 2, 2005. Available at http://library.ahima.org/xpedio/groups/public/documents/ahima/bok1_027397.hcsp?dDocName=bok1_027397.
4. White, Susan, and Sandra Nunn. “Two Educational Approaches to Ensuring Data Quality.” *Journal of AHIMA* 85, no. 7 (July 2014): 50–51.
5. American Health Information Management Association. “Embracing the Future: New Times, New Opportunities for Health Information Managers. Summary Findings from the HIM Workforce Study.”
6. Commission on Accreditation for Health Informatics and Information Management Education. “Welcome to CAHIIM.” Available at http://www.cahiim.org/index.html (accessed February 23, 2017).
7. Kwan, A. C. M. “Towards a Minimalist Approach to Lesson Planning of an Introductory Database Course.” *Proceedings of 2013 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)* (2013): 782–85.
8. Hylock, Ray, and Susie T. Harris. “Healthcare Database Management for Health Informatics and Information Management Students: Challenges and Instruction Strategies—Part 1.”
9. Ibid.
10. Connolly, Thomas M., and Carolyn E. Begg. “A Constructivist-based Approach to Teaching Database Analysis and Design.” *Journal of Information Systems Education* 17, no. 1 (2006): 43–53.
11. Kung, Hsiang-Jui, and Hui-Lien Tung. “A Web-based Tool for Teaching Data Modeling.” *Journal of Computing Science Colleges* 26, no. 2 (December 2010): 231–37.
12. Brusilovsky, Peter, et al. “Learning SQL Programming with Interactive Tools: From Integration to Personalization.” *Transactions on Computing Education* 9, no. 4 (January 2010): 19:1–19:15.
13. Kwan, A. C. M. “Towards a Minimalist Approach to Lesson Planning of an Introductory Database Course.”
14. Gudivada, V. N., J. Nandigam, and Yonglei Tao. “Enhancing Student Learning in Database Courses with Large Data Sets.” *37th Annual 2007 Frontiers in Education Conference: Global Engineering: Knowledge Without Borders, Opportunities Without Passports* (2007): S2D-13–S2D-17.
15. Yue, Kwok-Bun. “Using a Semi-realistic Database to Support a Database Course.” *Journal of Information Systems Education* 24, no. 4 (Winter 2013): 327–36.
16. Association for Computing Machinery. “Curricula Recommendations.” Available at https://www.acm.org/education/curricula-recommendations (accessed February 12, 2017).

17. Cohen, Eli B. "Rationale for the IRMA/DAMA 2000 Model Curriculum." In *Managing Information Technology in a Global Economy* (proceedings of the Information Resource Management Association International Conference). Hershey, PA: Idea Group, 2001, 612–616. Available at http://www.irma-international.org/proceeding-paper/rationale-irma-dama-2000-model/31683/.
18. Hylock, Ray, and Susie T. Harris. "Healthcare Database Management for Health Informatics and Information Management Students: Challenges and Instruction Strategies—Part 1."
19. Commission on Accreditation for Health Informatics and Information Management Education. "Welcome to CAHIIM."
20. Hylock, Ray, and Susie T. Harris. "Healthcare Database Management for Health Informatics and Information Management Students: Challenges and Instruction Strategies—Part 1."
21. Garcia-Molina, Hector, Jeffrey D. Ullman, and Jennifer D. Widom. *Database Systems: The Complete Book*. Upper Saddle River, NJ: Prentice Hall, 2002.
22. Date, C. J. *An Introduction to Database Systems*, 8th ed. Boston, MA: Pearson/Addison-Wesley, 2004.
23. Elmasri, Ramez, and Shamkant Navathe. *Fundamentals of Database Systems*, 6th ed. Boston, MA: Addison-Wesley, 2011.
24. Hoffer, Jeffrey A., Ramesh, V., and Topi, Heikki. *Modern Database Management*, 12th ed. Upper Saddle River, NJ: Prentice Hall, 2015.
25. Ullman, Jeffrey D., and Jennifer Widom. *A First Course in Database Systems*, 3rd ed. Upper Saddle River, NJ: Pearson/Prentice Hall, 2008.
26. Silberschatz, Abraham, Henry Korth, and S. Sudarshan. *Database System Concepts*, 5th ed. Boston, MA: McGraw-Hill Higher Education, 2006.
27. Kroenke, David M., and David J. Auer. *Database Processing: Fundamentals, Design, and Implementation*, 12th ed. Boston, MA: Pearson, 2012.
28. Kwan, A. C. M. "Towards a Minimalist Approach to Lesson Planning of an Introductory Database Course."
29. Yue, Kwok-Bun. "Using a Semi-realistic Database to Support a Database Course."
30. Teradata Corporation. "Teradata University Network." Available at http://www.teradatauniversitynetwork.com/ (accessed February 8, 2017).
31. Jukić, Nenad, Susan, Vrbsky, and Svetlozar Nestorov. *Database Systems: Introduction to Databases and Data Warehouses*. Boston, MA: Pearson, 2014.
32. Hoffer, Jeffrey A., Ramesh, V., and Topi, Heikki. *Modern Database Management*, 12th ed.
33. Watson, Richard. *Data Management: Databases and Organizations*, 5th ed. Hoboken, NJ: Wiley, 2006.
34. Gudivada, V. N., J. Nandigam, and Yonglei Tao. "Enhancing Student Learning in Database Courses with Large Data Sets."
35. Yue, Kwok-Bun. "Using a Semi-realistic Database to Support a Database Course."
36. Hylock, Ray, and Susie T. Harris. "Healthcare Database Management for Health Informatics and Information Management Students: Challenges and Instruction Strategies—Part 1."

Appendix A

Accompanying Data Sets

The full electronic appendix and data sets are available for download at http://blog.ecu.edu/sites/hsimcomputationallab/software/armc-datasets/.

Table 1

Background and Ability Comparison Matrix

Background	**Ability**	
	Nontechnical (N)	*Technical (T)*
Nontechnical (N)	*NN:* The student is prepared neither theoretically nor practically for the course. Many students with clinical or managerial backgrounds fall into this category. The difficulty here is ensuring that they do not fall behind early and/or become discouraged.	*NT:* The student possesses an aptitude for technology, but does not have the theoretical framework from which to draw. It is tempting to thrust advanced materials on these students, but their ability to perform a task does not prepare them for deductive processes (i.e., the ability to solve any problem as opposed to a concrete piece of a problem).
Technical (T)	*TN:* An unusual category, but possible for those who have taken more theoretical courses without necessarily being very hands-on. These students quickly grasp the theoretical components but need more applied instruction.	*TT:* The student has both the background and ability to progress through the course at an accelerated rate. Students in this category must be challenged without disturbing the overall grade distribution of the course. This topic is discussed further in the “Engaging the Advanced Student” section.

Table 2

Learning and Measurable Objectives by Construct Set for the Course Objective *Construct Conceptual Data Models (Conceptual Modeling)*

Learning Objectives	**Set**	**Example Measures (Quadrants for ECS)**
Entities		
- Strong and weak	MCS	- Appropriate use of entity types
- Aggregate and composite	ECS	- Appropriate use of advanced entity types (TN/TT)
Attributes		
- Basic types and key notation/construction	MCS	- Correct attribute and key notation - Correct primary key and foreign key construction
- Composite, multivalued, and derived	MCS	- Correct use and labeling of advanced data types
Relationships		
- Cardinalities	MCS	- Identify, assign, and justify minimum and maximum cardinalities (e.g., one-to-one/many, mandatory/optional)
- Degrees	MCS	- Identify relationship degree (e.g., unary, binary, ternary) - Construct relationship of appropriate degree
Supertypes and subtypes		
- Inheritance	ECS	- Identify inherited supertype attributes (TN/TT)
- Constraints (completeness and disjointness)	ECS	- Assign appropriate constraints (TN/TT)
- Discriminators	ECS	- Integrate subtype discriminators (TN/TT)
- Relationships	ECS	- Appropriate use of supertype/subtype relationships with external entities (TN/TT)

Abbreviations: ECS, extended construct set; MCS, minimum construct set; N, nontechnical; T, technical.
Notes: Quadrants listed for each ECS measure are defined in Table 1 and would be measured by a combination of the student survey and students' progress. The ECS and MCS for the objective in this example are available in Tables 3 and 4 of the following source (the previous article in this series).
Source: Hylock, Ray, and Susie T. Harris. "Healthcare Database Management for Health Informatics and Information Management Students: Challenges and Instruction Strategies—Part 1." *Educational Perspectives in Health Informatics and Information Management* (Winter 2016): 1–24.

Table 3

Learning and Measurable Objectives by Construct Set for the Course Objective *Implement Relational Databases via a RDBMS for Clinical Usage (Data Definition Language)*

Learning Objectives	**Set**	**Example Measures (Quadrants for ECS)**
Schemas	MCS	- Create and drop schemas
Tables	MCS	- Create, drop, and alter tables
Attributes	MCS	- Naming, data type selection, and inline constraints
Constraints		
- Primary and foreign keys	MCS	- Construct primary key and foreign key constraint - Identify attribute(s)/reference table included
- Optional, default values, and domains	MCS	- Construct constraints
Indices		
- Construction and defaults	MCS	- Create and drop basic indices - Default key indexing—DBMS-specific
- Composite, join, partial, and unique	ECS	- Create advanced indices (TT)
- Data structures (e.g., b-trees and bitmap)	ECS	- Assign most appropriate data structure by anticipated use and response requirements (TT)
Views		
- Basic views	MCS	- Create and drop basic views - Define view as inline-select and subquery replacement
- Temporary, updatable, and materialized	ECS	- Create and determine lifespan of temporary views (TT) - Create and drop updatable views (TT) - Determine view(s) to be materialized, create, and drop (TT)

Abbreviations: DBMS, database management system; ECS, extended construct set; MCS, minimum construct set; N, nontechnical; RDBMS, relational database management system; T, technical.
Notes: Quadrants listed for each ECS measure are defined in Table 1 and would be measured by a combination of the student survey and students' progress. The ECS and MCS for the objective in this example are in Tables 5 and 6 of the following source (the previous article in this series).
Source: Hylock, Ray, and Susie T. Harris. "Healthcare Database Management for Health Informatics and Information Management Students: Challenges and Instruction Strategies—Part 1." *Educational Perspectives in Health Informatics and Information Management* (Winter 2016): 1–24.

Table 4

Learning and Measurable Objectives by Construct Set for the Course Objective *Manipulate Clinical Data Using SQL—Access, Validate, and Integrate Clinical Data into a RDBMS (Data Manipulation Language)*

Learning Objectives	**Set**	**Example Measures (Quadrant[s] for ECS)**
SELECT/FROM/WHERE statements	MCS	- Single-table select - Filtering single-table results
Operators		
- Unary and ranges	MCS	- Filter data using simple comparison - Filter data using ranges
- Regular expressions	MCS	- Filter via regular expressions
Set operations		
- Basic: IN, NULL, DISTINCT, LIMIT	MCS	- Return distinct values - Limiting results - Filter data based on null values - Filter data based on another set
- Advanced: EXISTS, UNION, INTERSECT, EXCEPT	ECS	- Filter data based on attribute value existence (TN/TT) - UNION/INTERSECT/EXCEPT multiple sets (TT)
Aggregate Functions		
- Full sets	MCS	- Aggregate full set
- Subsets (GROUP BY/HAVING)	MCS	- Aggregate subset, necessitating the GROUP BY clause - Filter aggregate using the HAVING clause
Joins		
- Inner, natural, and Cartesian	MCS	- Inner/natural join to produce relational results - Understand Cartesian join as a way to identify missed join predicates (simply an error recognition tool in MCS)
- Outer and semi	ECS	- Outer joins for merging/converting data sets (TN/TT) - Semi-joins for data transmission over a network during the execution process (TT)
Sub-queries		
- Non-correlated	MCS	- Provide answer for a noncorrelated subquery
- Correlated	ECS	- Provide answer for a correlated subquery (TT) - Convert correlated subquery to statement/view structure (TT) - Rewrite correlated subquery (if possible) as a noncorrelated statement (TT)
Parentheses and SQL order of execution	MCS	- Appropriate use of parentheses to ensure correct logical order of predicate

		execution (e.g., Boolean parenthetical problem)
Creating tables via queries	MCS	- Create a table from the result of a query
Inserting, updating, and deleting tuples		
- Individual statements	MCS	- Insert, update, and delete tuples using single statements
- Via queries	MCS	- Insert, update, and delete tuples using SELECT results
Views		
- Basic views	MCS	- Integrate view(s) into a query - Use of view as inline-select and subquery replacement
- Temporary, updatable, and materialized	ECS	- Use a temporary (scoped) view (TT) - Update data via an updatable view (TT) - Use and refresh materialized views (TT)

Abbreviations: ECS, extended construct set; MCS, minimum construct set; N, nontechnical; RDBMS, relational database management system; T, technical.
Notes: Quadrants listed for each ECS measure are defined in Table 1 and would be measured by a combination of the student survey and students' progress. The ECS and MCS for the objective in this example are in Tables 7 and 8 of the following source (the previous article in this series).
Source: Hylock, Ray, and Susie T. Harris. "Healthcare Database Management for Health Informatics and Information Management Students: Challenges and Instruction Strategies—Part 1." *Educational Perspectives in Health Informatics and Information Management* (Winter 2016): 1–24.

Table 5

Learning and Measurable Objectives by Construct Set for the Course Objective *Administer a Secure Database System (Data Control Language)*

Learning Objectives	**Set**	**Example Measures (Quadrant[s] for ECS)**
Grant and revoke	MCS	- Grant/revoke schema, table, and sequence access - Update default privileges to allow/deny access to subsequently created objects
User- and role-based access controls	MCS	- Create user and grant/revoke at user level - Create role, grant/revoke at role level, and assign user to role
Commit and rollback	ECS	- Toggle auto-commit (TN/TT) - Manual commit and rollback (TN/TT)
Transaction blocks	ECS	- Use of transaction BEGIN and END/COMMIT commands to wrap multiple statements into a logical transaction (TN/TT) - Identify statements that cannot be executed in multistatement transactions (e.g., vacuum) (TN/TT)
Checkpoints	ECS	- Creating and restoring from checkpoints (TN/TT)
Views	ECS, MCS	- MCS: Describe view use in security - ECS: Create view to restrict data access (TT)

Abbreviations: ECS, extended construct set; MCS, minimum construct set; N, nontechnical; T, technical.
Notes: Quadrants listed for each ECS measure are defined in Table 1 and would be measured by a combination of the student survey and students' progress. The ECS and MCS for the objective in this example are in Tables 9 and 10 of the following source (the previous article in this series).
Source: Hylock, Ray, and Susie T. Harris. "Healthcare Database Management for Health Informatics and Information Management Students: Challenges and Instruction Strategies—Part 1." *Educational Perspectives in Health Informatics and Information Management* (Winter 2016): 1–24.

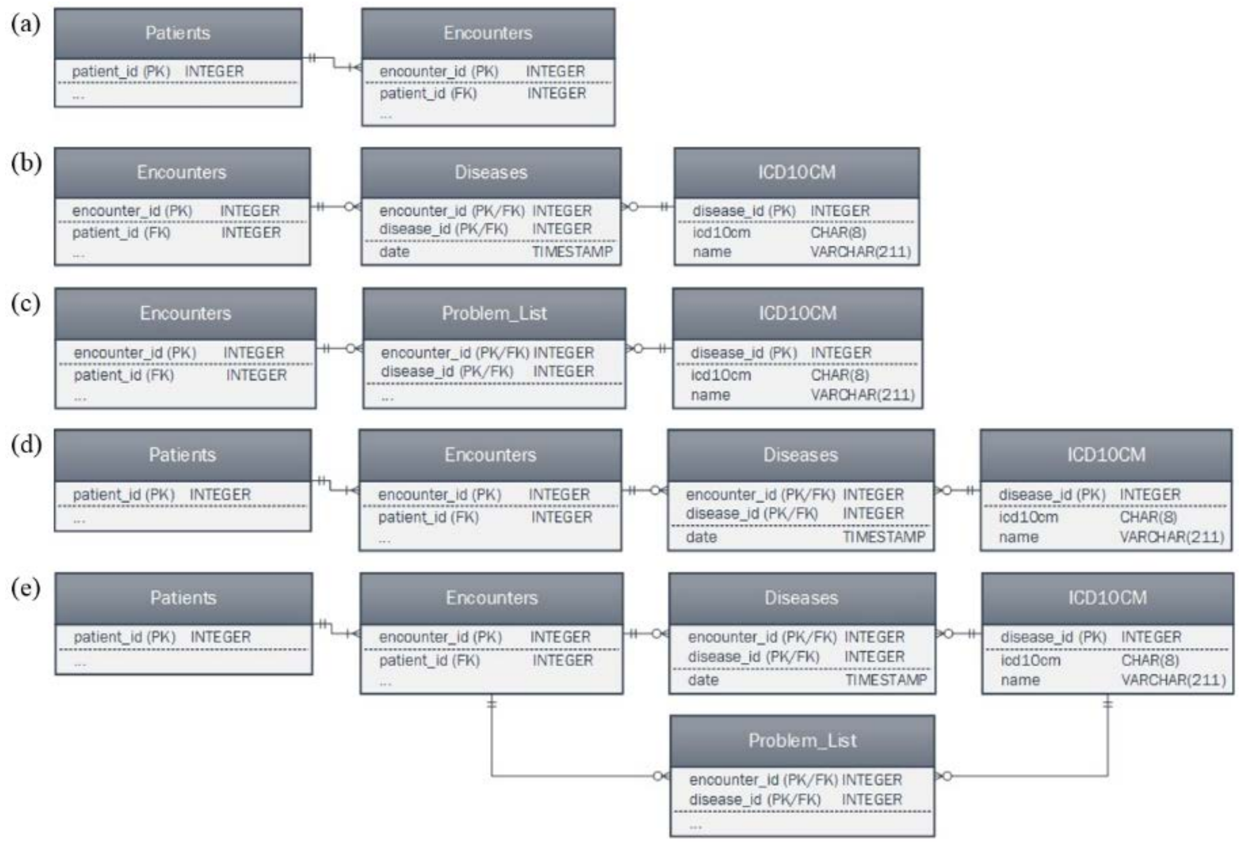
Table 6

Comparison of Approaches for Presenting Problems to Students

Approach	**Methodology to Create and Solve Problems**	**Ability of Instructor to Give Partial Credit**
All at once	Effectively all or nothing. Overwhelming, requires constant modification, results in a single point of failure.	Very difficult to give partial credit. (What do students know versus where did they get lost?) Fails to account for error propagation.
Incremental	Each piece builds on the previous piece, resulting in a single statement. Requires considerable preplanning by the student (nontrivial), and can result in a cascading failure.	Difficult to give partial credit. (Where did the student start and which step was constructed incorrectly?) Fails to account for error propagation.
Piecewise incremental	A piecewise approach with incremental substeps. Some problems are fully dependent on prior pieces. The pieces are nontrivially constructed by the instructor based on desired learning objectives. Point distribution determined by substep complexity.	Instructor determines the pieces, which makes it easier to give partial credit. Accounts for error propagation as long as the incremental substeps are easy to dissect.
Piecewise combination	Disjointed pieces with combination. The pieces are nontrivially constructed by the instructor based on desired learning objectives. Point distribution favors construction rather than combination.	Instructor determines the pieces, which makes it easy to give partial credit. Accounts for error propagation.

Figure 1

Data Modeling Example Solutions



Note: The parts of this example are as follows: (a) models patients and encounters, (b) diseases are assigned at an encounter, (c) shows problems and the encounters in which they were found, (d) combines (a) and (b), and (e) combines (c) and (d).

Figure 2

De-identified Patient Map Following the Safe Harbor Method

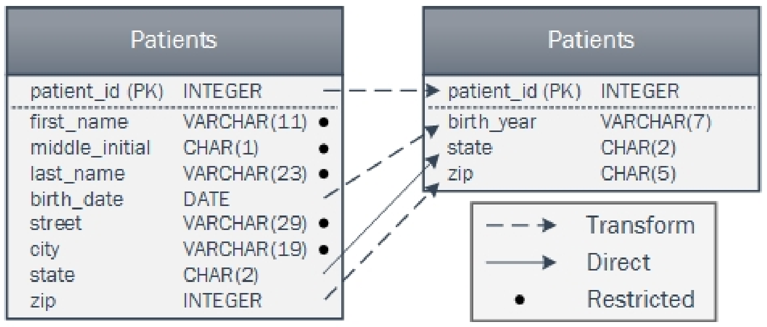


Figure 3

Procedure Assignment Conceptual Model for Error Recognition and Recovery Example 1

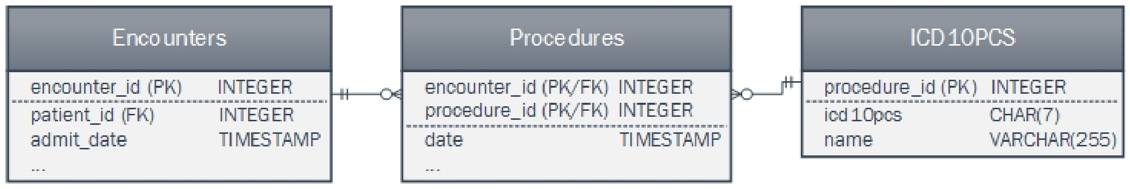
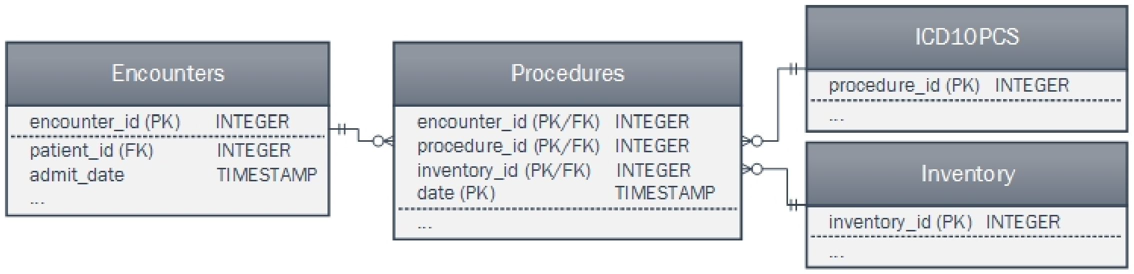


Figure 4

Procedure and Inventory Assignment Conceptual Model for Error Recognition and Recovery Example 2


Encounters
encounter_id (PK) INTEGER
patient_id (FK) INTEGER
admit_date TIMESTAMP
...
Procedures
encounter_id (PK/FK) INTEGER
procedure_id (PK/FK) INTEGER
inventory_id (PK/FK) INTEGER
date (PK) TIMESTAMP
...
ICD10PCS
procedure_id (PK) INTEGER
...
Inventory
inventory_id (PK) INTEGER
...