Automatic and Authentic eAssessment of Online Database Design Theory Assignments

No Author Given

No Institute Given

Abstract. One of the main reasons MCQ assessments are popular is that tests using text understanding is difficult and often erroneous. The emergence of large language models such as ChatGPT is not mature enough to help grading engineering assignments yet. Since MCQ tests are not well suited for summative assessment, scaling up eLearning for large number of students is difficult using non-MCQ tests. In this paper, we introduce a new eAssessment tool for database design courses that uses graphical conversations to understand learner's mental model of cognitive state. We show that the model is capable of substituting NLP for authentic assessment for eLearning.

Keywords: Intelligent tutoring, authentic assessment, eLearning, self-paced learning, user interface, graphical conversation, NLP substitute.

1 Introduction

Recent research suggest that among various types of technologies, e-learning is the most commonly validated mode of delivery, followed by m-learning, Learning Management Systems (LMSs), and social media services [8]. While technology adoption, LMS adoption in particular [12], in higher education is increasing significantly in the post-Covid era, barriers still exist that prevent effective use of these tools. These barriers largely include instructors' lack of confidence, and preparedness, and less than enthusiastic acceptance of the available tools [5]. It is, therefore, not surprising that instructor use these technologies mainly for blended learning [1] in which assessment[1] accounts only for 19% [16], perhaps because the instructors believe that the existing technologies are not a good fit for their style or personality [11], a factor that motivated the design of the system and research discussed in this article.

While there are tutoring systems for many CS subjects, there are significantly fewer systems for a first database course. The ones that are available, they split the topics into multiple tutoring systems (e.g, SQL [17], conceptual database design [9], relational algebra [10], and normalization [19]). We are, however,

[1] According to Rhodes et al. [16], the main usage of LMSs can be broken down in different categories as follows: Announcements (82%), Items (77%), Grade (71%), Folders (62%), Files (53%), Assignments (53%), Web links (30%), Plagiarism detection (22%), Discussion boards (21%) and Tests (19%).

unaware of a single comprehensive tutoring system for database teaching. Using multiple systems to teach a single subject introduces substantial impedance mismatch, management hurdles and potential equity issues. To address these complex set of issues, we have initiated *Project 360* as a comprehensive database tutoring system. It is designed to include four inter-connected and integrated tutoring systems – Conceptual Database Design (*CoDD*), Visual SQL (*ViSQL*), Relational Algebra Query Language (*ReliQ*), and Normalization and Database Design Theory (*NoDD*). In the remainder of this article, we discuss how NoDD approaches plagiarism immune authentic assessment and tutoring of functional dependency theory and database normalization.

2 Related Research

As opposed to a tutoring systems for teaching conceptual database modeling or database query languages, teaching functional dependency theory and database normalization probably are technically simpler. This is because these concepts are well defined and have established algorithms to compute them. As can be expected, there are several online normalization tools, or calculators, that do pretty well in computing various steps and components of dependency theories and normalization [15,18,19], and perform not too poorly in explaining the steps to the students. Unfortunately, that also means many existing online calculators, and more recently ChatGPT, make it extremely challenging to administer an online test without the risk of a dishonest student cheating.

From the point of view of tutoring the concepts of functional dependency theory and database normalization, the challenge is in designing a system that can get the point across, guide the students learn the theories, and improve learning overall. There are several online systems that actually are effective at varying degrees. When it come to assessment, to the best of our knowledge, there is probably no online system that covers authentic assessment. Even if one existed, we believe that would be hugely vulnerable to plagiarism and cheating and will go undetected. Our system *NoDD* is a first attempt to rectify the shortcomings of contemporary online database normalization tutoring and assessment systems.

There have been several few attempts at building database normalization tools. However, only a handful of them have been designed for tutoring [19], and none are designed for assessment even though it is an integral component of learning. Essentially, assessment is left as an offline exercise. Furthermore, among the online normalization calculators, only the Normalization Tool [19], Tool for Database Design [13], and Cho's normal form calculator [7] are live.

An effective normalization theory learning tool must introduce to the students the fundamental concepts of functional dependencies, the inference rules and cognate theories so that they are able to derive closure of a set of a dependencies (F^*). This fundamental understanding serves as the foundation for them to follow the notions of attribute closures (X_F^+), which can be used to compute candidate keys, covers, and to test loss-less join decomposition.

While there are a few tools that could potentially help compute some of these, most are not comprehensive, and often only cover a subset of computing needs. Even when they do, are not in a form that is suitable for a learning system such as ours. For example, for the set of functional dependencies $F = \{A \rightarrow CD, B \rightarrow DE, AG \rightarrow BC, AB \rightarrow G, BG \rightarrow A\}$ over the scheme $R = (ABCDEGH)$ the FD calculator developed by Chakravarty [6] does not compute the normal forms. While it computes attribute and functional dependency (FD) closures, and minimal covers, it computes everything exhaustively and unnecessarily without offering any insight or explanation. Similar concerns apply to the Tool for Database Design [13]. An interesting FD calculator by Cho [7] also generates explanations of all its derivations. However, it does all the computations in one go in a non-interactive fashion. Unfortunately it is also not being maintained.

However, none of these systems are open sourced and no support is offered. Furthermore, they are designed for online direct interactions by the users and no APIs are supported. Thus considering all the aspects, we have chosen to use the Normalization Tool [19] since it has a higher commitment to maintenance even though it does not offer all the functionalities we desire. It can optionally show the derivation steps, but without any explanations. Similar to FD Calculator, this is also modular and interactive. As we will highlight in the upcoming sections, NoDD also covers everything Normalization Tool does, and much more, but in a significantly unique way.

3 Database Design Assignments

Before we discuss the graphical conversation technique we have developed for eAssessment, it is perhaps helpful to discuss the nature and uniqueness of database design assignments, the subject of our eAssessment tool. One of the steps in database design is called schema normalization. A database scheme is a list of attributes, or column names of a table, and a set of functional dependencies that apply. To illustrate the concept of normalization, let us consider Ex 1 below.

Example 1. R is a relation scheme over attributes $\{A, B, C, D, E\}$, and associated functional dependencies $F = \{AB \rightarrow C, A \rightarrow C, C \rightarrow A, CD \rightarrow A\}$.

R is said to be non-3NF compliant, and a decomposition into a 3NF scheme is required. This is done so by discovering the canonical cover of the dependency set F, and following the 3NF decomposition algorithm yielding two schemes – $R_1(AC)$, and $R_2(ABDE)$.

The tools instrumental in this decomposition are concepts of Armstrong's Axioms and Inference Rules, candidate key identification, attribute closure X_F^+, canonical cover, loss-less join decomposition and 3NF decomposition algorithm. a 3NF decomposition of a scheme ensures that all functional dependencies are preserved and the splitting of the scheme is also loss-less. The decomposition of R into R_1 and R_2 above was accomplished first by determining that the dependency $CD \rightarrow A$ is redundant because $F \setminus \{CD \rightarrow A\} \models CD \rightarrow A$, i.e., even when $CD \to A$ is removed from F and $A \in CD^+_{F\setminus\{CD\to A\}}$. Then recognizing that $AB \to C$ is left-redundant. These two observations reduced F to the set $F_c = \{A \to C, C \to A$, and is called the canonical cover of F. We can also use standard techniques to discover the set of all candidate keys from either F or F_c easily. The complete set of candidate keys K of R is thus $ABDE$ and $BCDE$. Based on F_c and K, using the 3NF decomposition algorithm we obtain R_1 and R_2.

4 Graphical Conversational Interface

Our goal in the design of our graphical conversational interface for eAssessment is to construct a set of graphical alphabets or words with predefined meanings from which students will choose to write sentences with place holders for values. We then use a grammatical parsing system to understand the sentences to determine admissibility – or correctness. These sentences then can be assembled into a description of a solution. It turns out that the vocabulary can be compartmentalized based on the algorithms described in Sec 5.

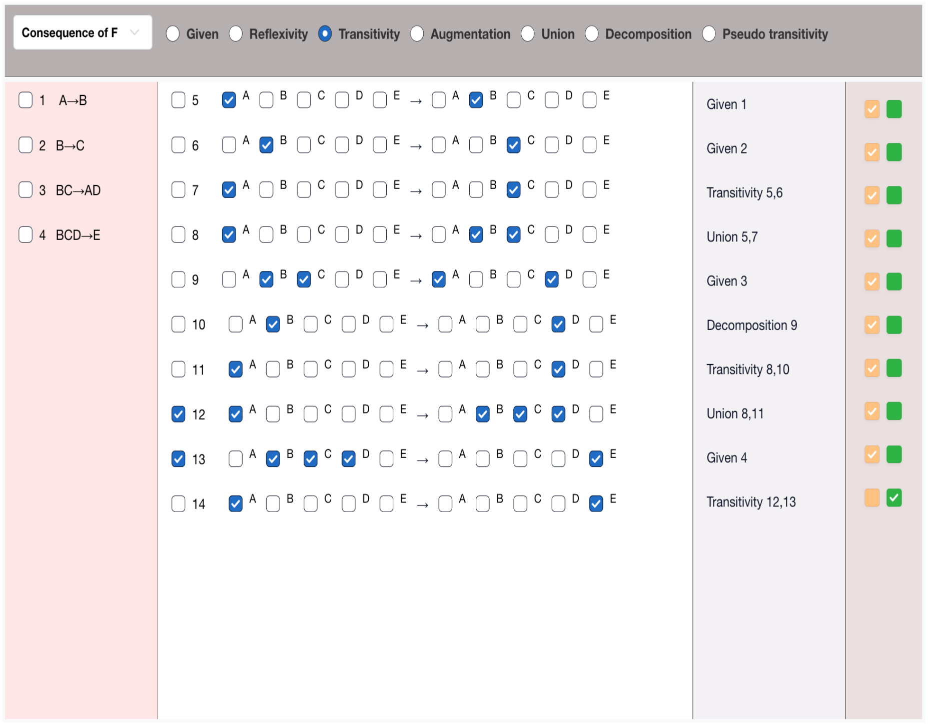
4.1 Language for Graphical Conversation

ChatBots, smart interfaces and many question answering systems use various forms of natural language conversations to engage with the users. While contemporary generative AI ChatBots actually construct responses live with a reasonable understanding of user questions, most other conversational systems are often simpler and all responses are predetermined. For example, the retired Expedia travel help ChatBot[2] was largely a keyword and short phrase comprehension based decision tree workflow execution engine that was capable of performing simple routine tasks such as cancelling a reservation, re-booking a travel, reimbursements inquiry, etc. Amazon's conversation is essentially a fixed dialogue system with deterministic steps and outcomes. The graphical, or visual conversation system we are considering is somewhat similar to the Expedia's keyword based ChatBot with markedly distinct features in which users make numerous decisions to drive the conversation, graphically withing a defined search space.

In NoDD, every conversation is based on a specific task, and the task defines the alphabet and the language for the conversation, all using graphical artifacts such as radio buttons, check boxes, selections, etc. As the Fig 1 shows, once the choice for Proof by Inference Rules is selected, the vocabulary and the conversational grammar is preset. In this case, the four FDs over the scheme R are the context, and the conversation is about derivation of inferences using the choices of rule application at the top (radio buttons). The sentences are the FDs, and the grammar for the construction of sentences are defined by these inference rules. NoDDs job is to ascertain if the user is constructing a valid sentence based on the grammar. It logs each step, so that feedback can be generated and the paragraph (the set of sentences) written can be graded. The editor in Fig 1 simply

[2] Expedia recently introduced ChatGPT based travel assistant.

allows the users construct fixed formatted sentences without offering too much help using check boxes. Only help it is offering is that it is constraining attribute choices within the scheme R[3].



Consequence of F — ○ Given ○ Reflexivity [✓ Transitivity] ○ Augmentation ○ Union ○ Decomposition ○ Pseudo transitivity

☐ 1 A→B	☐ 5	A → B	Given 1	✓ 🟩
☐ 2 B→C	☐ 6	B → C	Given 2	✓ 🟩
☐ 3 BC→AD	☐ 7	A → C	Transitivity 5,6	✓ 🟩
☐ 4 BCD→E	☐ 8	A → BC	Union 5,7	✓ 🟩
	☐ 9	BC → AD	Given 3	✓ 🟩
	☐ 10	B → D	Decomposition 9	✓ 🟩
	☐ 11	A → D	Transitivity 8,10	✓ 🟩
	☑ 12	A → BCD	Union 8,11	✓ 🟩
	☑ 13	BCD → E	Given 4	✓ 🟩
	☐ 14	A → E	Transitivity 12,13	✓

Fig. 1. NoDD graphical conversation for logical consequence of F.

4.2 NoDD Interface

The Normalization and Database Design tool we have designed is a unique and novel tutoring and authentic assessment system built for the sole purpose of database design theory teaching and learning. While there are several online calculator to help students solve database design theory related problems, none are comprehensive, and none can be used as an assessment tool. Technically, they cannot also be used as a tutoring tool, because they are designed to compute solutions directly without trying to help student to do it themselves and

[3] Removing this indirect help can be achieved by actually listing no attributes on both sides of the FDs, and letting the users write. For now, we believed that such an imposition does not buy much in terms of learning gains. Depending on user studies, this stance may change in the future.

removing conceptual blocks. The generic solution interface of NoDD, shown in Fig 1, can be used for all FD related problems with slight adaptation for both tutoring and assessment.

The interface is partitioned into four quadrants. The top horizontal quadrant runs from left edge to the right. This quadrant displays two types menus and options selection. Users choose to solve or practice one of eight problem types from a drop down list on the left top quadrant. In the middle, there are seven radio buttons to choose one of the six inference rules, or choose any of the FDs in F. To the right, there are three buttons – one to update the FD list with all the current edits, one to create a new solution panel, and one to conclude the solution. The remaining three quadrants are placed side by side just below the top quadrant. The leftmost quadrant lists all the active FDs with a serial number and a checkbox next to them. The middle quadrant is a workspace where users create solution steps using graphical alphabets. The rightmost quadrant primarily lists historical information of the solution steps. The sections below details how this interface is used to solve the eight problem types. For the sake of brevity, in this presentation, we will only discuss the use of NoDD in assessment mode. The tutoring and practice mode is functionally similar but has important differences in how feed backs are generated.

5 Functional Dependency Theory Algorithms in NoDD

Functional dependency theories are in the heart of database normalization. A sound understanding of its various components is essential for students to master normalization intricacies. Fortunately, all the concepts involved are algorithmically definable. In this section, we utilize those algorithms in the design of the conversational interfaces they must use to master functional dependency concepts and to help students learn normalization theory using NoDD.

5.1 Logical Consequence of FDs

Logical consequences of a set of dependencies F can tested in two principal ways – by derivation using Armstrong's axioms and inference rules, and by attribute closure computation. In both of these approaches, our principal goal is to understand if the student is following the correct algorithm by reconstructing it. Figure 1 illustrates how the the process works in NoDD in reference to Ex 2.

Example 2. Let the relational scheme be $R = (ABCDE)$ and the set of dependencies be $F = \{A \rightarrow B, B \rightarrow C, BC \rightarrow AD, BCD \rightarrow E\}$. Show that $F \models A \rightarrow E$.

Derivation Method The graphical conversation proceeds as follows: To begin, the student selects the set of dependencies F, the scheme R, and the method he will use, and NoDD immediately prepares the interaction process as shown in Fig 1 for $F \models f$ using derivation. NoDD displays the four dependencies of F

and assigns the FDs a chronological number 1 through 4. Since a derivation is chosen, a new FD is expected, and a new blank FD with a number 5 is displayed in the middle quadrant as shown below:

$$\square\ f_5 : \square^A\square^B\square^C\square^D\square^E \to \square^A\square^B\square^C\square^D\square^E \text{ – Transitivity 1 and 2 } \square\ \square$$

This is the result of the student choosing first "Transitive" in the top panel indication he is applying transitivity between FDs 1 and 2 (shown checked in the second quadrant in the left. The right quadrant comment shows what rule being applied between the FDs 1 and 2. The green checkbox is checked if the student wishes to end the derivation process indicating he believes the proof is complete.

However, the derivation is not yet complete since the student is yet to construct the derived dependency after applying transitivity between FDs 1 and 2, which should be $A \to C$. Student uses the check boxes on both sides of the dependency template f_5 to pick the attributes of R he believes should be in this new dependency. Figure 1 also shows the proper and complete derivation of the proof for $F \models A \to E$. Notice that, at every step until f_{14}, the yellow checkbox was selected to indicate that the derivation is incomplete at this step and a new template is needed for the next derivation. Finally the green checkbox was chosen at f_{14} to indicate completion of the proof.

Attribute Closure Method When the "Attribute Closure" method is chosen, an interface simulating the derivation process below is presented. It starts by instantiating a derivation template as shown below:

$$\{A\}^+_F = \{\square^A\square^B\square^C\square^D\square^E\} \text{ – FD } \square\ \square$$

Here too, the derivation follows systematically until the green checkbox is selected. The complete derivation and proof by attribute closure is shown below. In this process, students choose one dependency at each step that he believes is applicable, and adds or deletes the set of attributes believed to be in the closure.

$$\{A\}^+_F = \{☑^A\square^B\square^C\square^D\square^E\} \text{ – Reflexivity } ☑\ \square$$
$$\{A\}^+_F = \{☑^A☑^B\square^C\square^D\square^E\} \text{ – FD 1 } ☑\ \square$$
$$\{A\}^+_F = \{☑^A☑^B☑^C\square^D\square^E\} \text{ – FD 2 } ☑\ \square$$
$$\{A\}^+_F = \{☑^A☑^B☑^C☑^D\square^E\} \text{ – FD 3 } ☑\ \square$$
$$\{A\}^+_F = \{☑^A☑^B☑^C☑^D☑^E\} \text{ – FD 4 } \square\ ☑$$

Grading Student Responses In both methods, the selections, rule applications, and the derivations are systematically logged as shown in the fourth quadrant history log. It is a simple process to determine if the student is incorrect, and all NoDD has to do is check if the final step is reached correctly. Although we are not discussing NoDD's tutoring mode in this article, it is not too hard to see how NoDD could offer feedback to the student in the event he made a mistake, which NoDD is able to detect at each step. However, designing effective hints is a research topic by itself, and will be presented separately.

5.2 Candidate Keys

One of the most important steps in database design is the discovery of candidate keys. However, to be able to determine if a relation scheme is in Third Normal Form (3NF) or Boyce-Codd Normal Form (BCNF), it is necessary to discover all candidate keys, not just one. Therefore, mastering a systematic derivation technique is essential. It turns out that candidate keys can be derived in three principal ways – exhaustive or iterative method, heuristic method and FD based reduction method. We present the tools we have developed for each below.

Exhaustive Method In this method, a student basically tries every possible combination of the attributes in the scheme, and attempts to determine if the attribute closure recovers the entire scheme to declare the combination a superkey or a candidate key. The process is similar to the technique outlined in Sec 5.1 for attribute closure computation. NoDD prepares a derivation template of attribute closure for all possible combinations, along with four check boxes – yellow, green, cyan and red, as shown below. As usual, yellow requests another step in the derivation process. Checking green indicates the current attribute set is a candidate key, cyan declares it a superkey, and red concludes it to be a non key, and the derivation for this combination stops.

$$\{A\}^+_F = \{\square^A\square^B\square^C\square^D\square^E\} \text{ – FD 1 } 🟨 🟩 🟦 🟥$$

This method is quite laborious and demands significant attention. A better method is the heuristic method described next that eliminates a large number of combinations that need to be tried, and thus shortens the process.

Heuristic Method A sound heuristic that can be used to determine candidate keys involves separating the attributes into three distinct sets called the K^+, K^- and $K^?$. The attributes that appear on the left hand side always in any FD are placed in K^+ because they will always be in any candidate key. Similarly, the attributes that always appears on the right hand side, cannot appear in any candidate key since they are determined by other attributes, and thus belong to the K^- set. The remaining attributes are placed in the $K^?$ set because no decisions can be made about their status. The idea is to try attribute closures of all the combinations of the sets attributes in $K^+ \cup K^?$, and not include attributes in K^- at all.

The derivation step follows the process in Sec 5.2 preceded by the identification of these three sets, and displays only the combinations involving the sets identified in K^+ and $K^?$. In Ex 2, only E is on the right hand always, and the remaining attributes are on both sides. Therefore, the identification follows the step below:

$$K^+ = \{\square^A\square^B\square^C\square^D\square^E\} \text{ – FD } ☑$$
$$K^- = \{\square^A\square^B\square^C\square^D☑^E\} \text{ – FD 4 } ☑$$
$$K^? = \{☑^A☑^B☑^C☑^D\square^E\} \text{ – FD 1, 2, 3, 4 } ☑$$

NoDD therefore will list all combinations of $ABCD$ from the lowest cardinality combinations to the highest, and allow the process of key identification as in Sec 5.2. Figure 5.2 also shows the derivation process of the candidate keys.

Reduction by Elimination Method The elimination method though simple in concept, it is a complicated procedure to implement on a computer screen. It is basically a tree building process with the entire scheme of the relation at the root. Since all schemes are at least a superkey by itself, it can be used as a starting point for the reduction process. The idea is to use an FD in F to and remove the attributes (say X) at the right hand side of it from the scheme at the root. This is because, X can still be recovered we the attribute closure of $R \setminus \{X\}_F^+$ is computed, and yet the scheme, or the superkey is reduced to a smaller superkey. The goal is to find the smallest superkey, the candidate key, by repeated applying this technique until no more attributes can be removed.

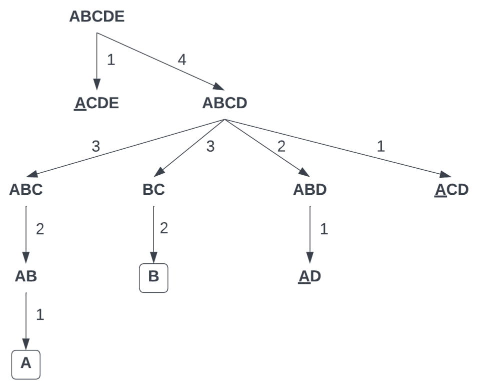


Fig. 2. Partial derivation of candidate keys using elimination method.

Fig 2 shows a partial expansion of the scheme R over the FDs F. The numbers on the arrows show the FD application. For example, using FD 1, the attribute B was removed from the root to create the node $ACDE$. The elimination process in this branch stopped because no other FD's left hand side is a subset of $ACDE$ to help us eliminate any attribute. Though it is a terminal node, it is not minimal as we will see later. At the left tallest branch, we find a terminal node A (marked inside a box) which is minimal and was reached via repeated FD application in a similar way. Another boxed and terminal node is B. The other terminal nodes now can be checked to see if they are super sets of a boxed or any other terminal nodes and thus cannot be considered a candidate key. In this partial tree, all terminal non-boxed nodes are super sets of A.

This complex tree expansion and attribute elimination process is modeled in NoDD in a level wise fashion within the interface we have used so far. To initiate the elimination process, NoDD presents the root node as

$1:\ \{ABCDE\}$ – FD 🟨☐ 🟩☐ 🟦☐ 🟥☐

On clicking on the yellow checkbox, it generates the following display, where in the left is the root node, and at the right is a child of the root node at level 2.

$1:\ \{ABCDE\}$ – FD 🟨☐ 🟩☐ 🟦☐ 🟥☐ $2:\ \{\Box^A\Box^B\Box^C\Box^D\Box^E\}$ – FD 🟩☐ 🟦☐

To deduce node $ACDE$ as shown in Fig 2, the user chooses FD 1 at the left, checks B on the right node number 2. He also checks if this is a candidate key (green box), or is a superkey (cyan box). In this case, the user chose the cyan box, when the display reduces to the presentation below.

$1:\ \{ABCDE\}$ – FD 🟨☐ 🟩☐ 🟦☑ 🟥☐
$2:\ \{ACDE\}$ – FD 1 🟨☐ 🟩☐ 🟦☑ 🟥☐

When the user selects the yellow box again on node 1, the numbering of the nodes changes and the following node structure is presented.

$1:\ \{ABCDE\}$ – FD 🟨☐ 🟩☐ 🟦☑ 🟥☐
$2.1:\ \{ACDE\}$ – FD 1 🟨☐ 🟩☐ 🟦☑ 🟥☐ $2.2:\ \{\Box^A\Box^B\Box^C\Box^D\Box^E\}$ – FD 🟩☐ 🟦☐

On expansion in a similar manner, the structure below is generated.

$1:\ \{ABCDE\}$ – FD 🟨☐ 🟩☐ 🟦☑ 🟥☐
$2.1:\ \{ACDE\}$ – FD 1 🟨☐ 🟩☐ 🟦☑ 🟥☐
$2.2:\ \{ABCD\}$ – FD 4 🟨☐ 🟩☐ 🟦☑ 🟥☐

A complete expansion of the derivation tree will identify the two candidate keys A and B. The final node showing A as the candidate key will be presented as

$2.2.1.1.1:\ \{A\}$ – FD 1 🟨☐ 🟩☑ 🟦☐ 🟥☐

Grading In this step too, grading is by retracing the path the student followed and making sure that the steps followed are correct.

5.3 Covers of Dependencies

Covers of dependencies require two sets of dependencies, say F and G. To establish the fact that F covers G, we are required to show that $\forall g(g \in G, F \models g)$. However, to show that $F \equiv G$, we must show that $\forall f, g((f \in F, G \models f) \wedge (g \in G, F \models g))$. For this task, we again repurposed the interface discussed in Sec 5.1. This time, instead of one set of dependencies, we list two sets, and allow a checkbox to make one of them active, say F. The required step here is that the student must choose (the system does not require it) an FD $g = X \rightarrow Y \in G$ by checking the box next to it. Once he does, a dialog box appears as before in a slightly different manner as shown below:

$$\{\square^A\square^B\square^C\square^D\square^E\}^+_F = \{\square^A\square^B\square^C\square^D\square^E\}$$ – FD 1 🟨 🟩 🟥

This time, we require the user to choose the set of attributes (X) in the left of the equal sign for which the attribute closure is to be computed. Then follow the process in Sec 5.1 with respect to the dependencies in F and demonstrate that he could recover Y in the right side of the equal sign. If he could not, he should check the red box and declare that F does not cover G. Instead, if $Y \subseteq X^+_F$, then he should choose to check the green box. Then repeat the same process for all the dependencies in G. Every time he checks the green box, the check box next to the FD in G also turns green, red otherwise. To show that $F \equiv G$, he must now switch F to G as the active FD set, and show that $G \models f$, for all $f \in F$.

5.4 Canonical Cover of FDs

Theoretically, the canonical cover of a set of FDs F is the set of FDs F_c such that there is no redundant FDs in F_c and no FDs has extraneous attributes in their left hand sides, i.e., minimal. To compute F_c, a student must follow the following three steps, in no particular order.

Decomposing FDs While decomposition of FDs of the form $f = X \to Y$ into one attribute in the right hand side is not necessary, but doing so makes the remaining process simpler. However, some form of FD union operation may become necessary to simply decomposed schemes, again not mandatory. This step can be achieved by choosing an FD, and clicking a button to decompose it. Once clicked, NoDD decomposes the FD into $X \to A$ form, and reorders and renumbers the FDs. The set of FDs generated to replace the FD $X \to Y$ is equal to the number of distinct attributes $A \in Y$. For example, the FD $BC \to AD$ is replaced by $BC \to A$ and $BC \to D$ using the simple application of the decomposition rule.

Redundant FD Removal An FD $f = X \to A \in F$ is redundant, if F still implies f, i.e., $F \setminus \{f\} \models f$. That means, for an FD $f = X \to A$ to be non-redundant, $A \notin X^+_{F\setminus\{f\}}$ must hold. We again re-purpose the cover computation subsystem discussed in Sec 5.3 with the following adjustments. To test if an FD is redundant, the students selects an FD f from the FD list F by checking the box next to it. He is then presented with the conversation below:

$$\{\square^A\square^B\square^C\square^D\square^E\}^+_F = \{\square^A\square^B\square^C\square^D\square^E\}$$ – FD 1 🟨 🟩 🟥

and he must follow the procedure discussed in Sec 5.3. At the end, he must check either ”not redundant” (green), or “redundant” (red). Once checked, the button next to the FD in FD list is colored green or red, and the FD is unchecked. However, he can optionally check the FD checkbox again to remove it from the FD list, when it is greyed out. The process can be repeated for all the FDs in F.

Removing Left-Redundancies in FDs To remove left redundancies, we use the same interface and for redundant FD removal, but now turn on the left-redundancy removal radio button. Thus the multi-function red button is able to allow removal of attributes from the FD currently checked. This time, check boxes in the attributes in the FD appears, and students are able to select the attributes they wish to remove. The checkbox next to the FD stays unchecked. This time too, if the FD checkbox is selected, the attributes checked in the FD is greyed out to indicate removal.

5.5 Normal Form Decomposition

During this step, students choose a set of dependencies F, and use the tools described in Secs 5.2, and 5.4 to compute all the candidate keys K of R and the canonical cover F_c of F. While there are definitions for both 3NF and BCNF to test is a scheme is already in these normal forms, and by checking those conditions we could potentially save some time and avoid unnecessary decomposition, for the sake of brevity, we will not address this issue in this paper. In the following two sections, we present the process a student will follow to decompose a scheme into 3NF or BCNF assuming that the scheme warranted the decomposition[4].

Third Normal Form Recall that the 3NF status of a scheme is related to transitive and partial dependence of attributes on non-key attributes. To avoid making mistakes by following a manual inspection, the algorithms for these two normal forms ensure a correct decomposition. The algorithm leverages the canonical cover computation and asks to create a distinct scheme S using the attributes in a FD $f \in F_c$ only if $S \not\subseteq R_i$ where R_i, $1 \leq i \leq n$, is one of the decomposed schemes already created, and then renaming S as R_{n+1}. Finally, if none of the R_is is a super set of of any of the candidate keys in K, then create a final distinct scheme R_{n+1} using the attributes of one of the keys in K.

To respond to a test question on 3NF decomposition, students can start by computing the canonical cover and the candidate keys. Or, they can enter them in the system manually. Technically, the interface will have a set of dependencies and a set of keys as shown in Fig 5.5. Students identify an FD or a key by checking the box, to work on it. NoDD presents the scheme as shown below that is tied to the FD or key selected. Selecting an FD displays

R_1 : $\{\square^A\square^B\square^C\square^D\square^E\}$ – FD 1 🟩 🟥

and the student selects the attributes for the decomposed scheme. On the other hand, selecting a key displays

R_1 : $\{\square^A\square^B\square^C\square^D\square^E\}$ – Key 1 🟩 🟥

[4] It must be noted here that even if the scheme is already in one of these normal forms, attempting to decompose it or doing so does not render the decomposition unsound, or introduce a design error. Finally, most often then not, the algorithms will return the original schemes anyway.

and the scheme is created as in the previous case. Once the green box is checked, the scheme is added as one of the decomposed schemes. Checking red box removes it from the set. We should note that we prefer not to aid in constructing the schemes by pre-selecting the attributes based on the FD, key of the principles we have discussed. This will defeat the purpose of testing the cognitive process of the students. We offer them enough so that they are able to pick the right statements, not responses.

Boyce-Codd Normal Form BCNF decomposition if a complicated and involved algorithm. The algorithm fundamentally is a binary tree expansion algorithm. We, therefore, adapt an interaction system similar to candidate key discovery algorithm discussed in Sec 5.2. This time, we expect a right skewed binary tree, though we support a general tree expansion as before. Therefore, we simplify the numbering scheme for node identification with the following changes. The root node is displayed as follows along with its first possible child:

1 :	$\{ABCDE\}$	🟨	🟩
a :	$AB \rightarrow C$	☐	
b :	$A \rightarrow C$	☐	
c :	$CD \rightarrow A$	☐	
K_1 :	ABDE	☐	
K_1 :	ACDE	☐	

2.1 :	$\{\square^A\square^B\square^C\square^D\square^E\}$	– FD	🟩
a :	$\square^A\square^B\square^C\square^D\square^E \rightarrow \square^A\square^B\square^C\square^D\square^E$	🟨	
K_1 :	$\square^A\square^B\square^C\square^D\square^E$	🟨	

Students are expected to choose one of the dependencies in the root node, and derive a child node based on the dependency just chosen. In this child node, the student also is expected to identify the FDs and keys that would hold. He uses the yellow check boxes to add one more dependency or key, and the green box to confirm the node and exit. By checking the attribute boxes, he constructs the scheme, FDs and keys.

Finally, once the child green box is checked, a new child node is created along the information just supplied. Student can click on the parent's yellow box again to generate one more template for a new child node, and new description begins. This time, the child is assigned a node number 2.2, since there is already a number 2.1. Until the green box of the root or parent is checked, more children can be generated. Furthermore, when node 2.1 (say) is used as the root node, the new child is numbered 3.1 (child of a level 2 node), and all subsequent level three nodes are sequentially numbered. Figure 5.5 shows the complete expansion of the BCNF decomposition of the scheme in Ex 1.

6 Conclusion and Future Research

We believe there are two principal issues in authentic eAssessment we need to tackle. First, we need to use a language that is easily and correctly interpretable for the student to express his mental model of the problem being solved or the question he is responding to. Using a natural language as a modality of communication, such as English, is ideal in the age of large language models (LLM) and seems appropriate. However, as a recent conversation[5] with ChatGPT reveals, it is not that simple and it is far less perfect than one might expect. Leaving the costs of developing tailored technologies using LLMs as backbones aside, its efficacy and efficiency of human AI communication for such purposes require further improvement.

Excellent alternatives to natural language based human computer communication are visual languages [2,4], and visual interfaces [14]. Visual interfaces are finite and simpler than visual languages, while visual languages are more open and difficult to design, and thus limits its scope. In this paper, we blended these two technologies into a language to capture the solution process a student follows in answering database design questions. Though it is limited in one application, it is highly focused and effective. In particular, it is easy to follow the chain of thought and easy to interpret the thought process for the purpose of grading.

Second, the approach we have adopted helps us stay plagiarism averse. Given the nature of the subject and the availability of a large number of online tools that can instantly provide the answers to all test questions, we needed to focus on the capturing the analytical process followed by the test takers that they cannot glean from the internet calculators. In the absence of a proctor in online test taking and the real possibility of cheating, forcing the student to display the depth of subject matter understanding and comprehending the explanations is a critical issue. NoDD achieves that goal by offering a subject-grounded conversational language for question answering and interpreting the responses flawlessly.

In this paper, we refrained from discussing the algorithms for interpreting the responses for the assessment and feedback generation during tutoring sessions [3, 20] . Though it is easy to sense that the NoDD interface diligently collects the sequence of thought process of a student, interpreting them requires careful attention to details. Feedback generation in the form of prompts and suggestions just enough to put the student on track also requires careful design and sophisticated models. These are some of the issues we seek to explore in our future research.

References

1. B. Anthony Jr, A. Kamaludin, A. Romli, A. F. Mat Raffei, D. Phon, A. Abdullah, and G. Ming. Blended learning adoption and implementation in higher education: A theoretical and systematic review. *TKL*, pages 531–578, 06 2022.
2. P. Ayala, I. Barandiaran, D. Vicente, and M. Graña. Exploring simple visual languages for real time human-computer interaction. In *IEEE VECIMS 2003, Lugano, Switzerland, 27-29 July 2003*, pages 107–112, 2003.
3. M. A. Ball and D. D. Garcia. Autograding and feedback for snap!: A visual programming language (abstract only). In *ACM SIGCSE 2016, Memphis, TN, USA, March 02 - 05, 2016*, page 692, 2016.
4. P. Bottoni, M. F. Costabile, S. Levialdi, M. Matera, and P. Mussio. Principled design of visual languages for interaction. In *IEEE VL 2000, Seattle, Washington, USA, September 10-13, 2000*, pages 145–155, 2000.

[5] https://chat.openai.com/share/04aaf386-a76b-4ad7-b814-d840d139eb0b

5. C. Buabeng-Andoh. Factors influencing teachers' adoption and integration of information and communication technology into teaching: A review of the literature. *IJEDICT*, 8:136–155, 01 2012.
6. A. Chakravarty. Functional dependencies checker. https://tinyurl.com/yrzrhcm3, 2018. Accessed: 8/1/2022.
7. R. Cho. Tool for database design. https://tinyurl.com/45jz7bpv, 2017. Accessed: 8/1/2022.
8. A. Granic. Educational technology adoption: A systematic review. *Education and Information Technologies*, 04 2022.
9. N. Jukic, S. Vrbsky, S. Nestorov, and A. Sharma. Erdplus. https://erdplus.com/, 2020. Accessed: 7/31/2022.
10. J. Kessler, M. Tschuggnall, and G. Specht. Relax: A webbased execution and learning tool for relational algebra. In *BTW 2019, 4-8 März 2019, Rostock, Germany*, volume P-289 of *LNI*, pages 503–506, 2019.
11. J. Kimmerl. Does who am I as a teacher matter? exploring determinants of teachers' learning management system adoption in education style. In *ICIS 2020, Hyderabad, India, December 13-16, 2020*. Association for Information Systems, 2020.
12. M. N. Ngafeeson and Y. Gautam. Learning management system adoption: A theory of planned behavior approach. *IJWBLT*, 16(1):27–42, 2021.
13. U. of Illinois Springfield. Relational database tools. https://tinyurl.com/3xhvxkby, 2018. Accessed: 8/1/2022.
14. M. Ohira, H. Masaki, and K. Matsumoto. CICRO: an interactive visual interface for crowd communication online. In *OCSC 2011 at HCI 2011, Orlando, FL, USA, July 9-14, 2011*, volume 6778 of *LNCS*, pages 251–260. Springer, 2011.
15. H. I. Piza-Dávila, L. F. G. Preciado, and V. H. Ortega-Guzmán. An educational software for teaching database normalization. *Comput. Appl. Eng. Educ.*, 25(5):812–822, 2017.
16. J. Rhode, S. Richter, P. Gowen, T. Miller, and C. Wills. Understanding faculty use of the learning management system. *Online Learning Journal*, 21:68–86, 01 2017.
17. U. Röhm, L. Brent, T. Dawborn, and B. Jeffries. SQL for data scientists: Designing SQL tutorials for scalable online teaching. *VLDB*, 13(12):2989–2992, 2020.
18. C. Stefanidis and G. Koloniari. An interactive tool for teaching and learning database normalization. In *Proceedings of the 20th Pan-Hellenic Conference on Informatics, Patras, Greece, November 10-12, 2016*, page 18. ACM, 2016.
19. J. Wang and B. Stantic. Facilitating learning by practice and examples: a tool for learning table normalization. In *BCI 2019, Sofia, Bulgaria, September 26-28, 2019*, pages 35:1–35:4. ACM, 2019.
20. C. Watson, F. W. B. Li, and R. W. H. Lau. Learning programming languages through corrective feedback and concept visualisation. In *ICWL 2011, Hong Kong, China, December 8-10, 2011*, pages 11–20, 2011.