ERDoc: A Web Interface for Entity–Relation Modelling

Matías López
DCC UChile
Santiago, Chile
mlopez@dcc.uchile.cl

Sebastián Ferrada
DCC & IDIA UChile. IMFD Chile
Santiago, Chile
sebastian.ferrada@uchile.cl

Aidan Hogan
DCC UChile. IMFD Chile
Santiago, Chile
ahogan@dcc.uchile.cl

ABSTRACT

Computer Science professors at the University of Chile have found that their students often lack the abilities required to adequately model a relational database, for example, in the context of capstone Software Engineering projects. These students have passed an introductory Databases course covering conceptual modelling via Entity–Relation (ER) diagrams. A possible cause is that modelling tools found on the Web force students to immediately think in terms of tables and foreign keys instead of in terms of concepts and how they connect. In this paper, we present ERDoc: an application aimed at assisting students and other users through the modelling process. We define a syntax to allow users to describe entities, their attributes and relationships; from these descriptions, we automatically and dynamically generate ER diagrams. Both syntactic and semantic errors are detected and informed to the user. Preliminary evaluations with professors, teaching assistants and students show that ERDoc Playground is usable and useful in the task of conceptual modelling of databases, and thus has the potential to improve the conceptual-modelling aptitudes of Computer Science students.

CCS CONCEPTS

• **Information systems → Entity relationship models**; *Markup languages*; *Web applications*.

KEYWORDS

Entity-Relationship Diagrams, Relational Database Modelling, Teaching in Databases

ACM Reference Format:
Matías López, Sebastián Ferrada, and Aidan Hogan. 2024. ERDoc: A Web Interface for Entity–Relation Modelling. In *3rd International Workshop on Data Systems Education: Bridging education practice with education research (DataEd '24), June 9, 2024, Santiago, AA, Chile.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3663649.3664372

1 INTRODUCTION

Entity-Relationship (ER) Diagrams are useful for the conceptual modelling of databases, especially in Computer Science curricula [15]. ER diagrams allow the definition of the relevant objects being modelled along with how they interrelate. A high-quality ER diagram often leads to a well-designed relational schema in 3NF [9].

DataEd '24, June 9, 2024, Santiago, AA, Chile

ACM ISBN 979-8-4007-0678-3/24/06
https://doi.org/10.1145/3663649.3664372

Computer Science professors at the Universidad de Chile have found that students lack conceptual modelling skills. We believe that this problem is due to *a*) students tending to immediately think in terms of tables instead of entities, *b*) students not seeing the benefits of conceptual modelling, and *c*) a lack of publicly available and open source tools for conceptual modelling when compared with tools available for designing physical models [5], such as *dbdiagram* [1]; MySQL Workbench [2]; pgModeler [3]; etc.

In this paper, we introduce ERDoc Playground: a web application that allows users to dynamically generate Entity-Relationship (ER) diagrams by specifying them with a custom language we call ERDoc. ERDoc Playground supports basic and advanced features of ER Diagrams, as specified by Chen [6], namely entities, *n*-ary relationships, and weak entities. Other features like class hierarchies and aggregation [14] are also supported. Furthermore, ERDoc Playground displays the diagrams in different notations and automatically places the visual elements in a layout. We evaluate ERDoc Playground on its performance and usability. Results show that ERDoc Playground performs well with reasonably large diagrams (hundreds of entities), and users find it intuitive and useful.

Before describing our markup language in Section 3 and ERDoc Playground itself in Section 4, we review solutions for database modelling in Section 2. We present our evaluation of ERDoc Playground in Section 5 and give our concluding remarks in Section 6.

2 CURRENT SOLUTIONS

Students and other practitioners have a number of options for modelling ER diagrams, which we divide into two categories.

Generic drawing solutions. Generic drawing tools – such as Diagrams, `draw.io`, LaTeX/tikz, Paint, Powerpoint, Visio, or even freehand sketching – can be used to draw ER diagrams using Chen notation [6]. While a simple and effective solution, we notice that students tend to spend a lot of time concerned about the minutiae of the graphical notation, such as the dashed lines, double lines, arrows, etc. Furthermore, no automated feedback is provided.

Modelling solutions. Other options are rather tailored for modelling tasks, such as creating ER diagrams. Carvalho et al. [5] provide a survey of 20 highlighted tools selected from more extensive online lists.[1] Some of these tools are commercial, while others are free. Some are desktop applications, while others are web applications. Surprisingly, Carvalho et al. [5], in reference to the ER model, note that "*Despite this graphical representation, the physical data model produces a better comprehension of the relationships*". Perhaps for this reason, systems that claim to provide functionality for "ER diagrams" rather support diagramming relational models, thus conflating conceptual and physical models. In these applications, one can find tables, attributes, foreign keys, private keys, etc.

[1] https://www.databasestar.com/data-modeling-tools/

No graphical distinction is made between entities and relations. While perhaps sufficient for more experienced database practitioners who wish to design and/or document a relational schema, we claim that such systems, by bypassing conceptual modelling, do not help students to learn good modelling practices [9].

Conceptual Modelling in Education. Davis [7] designs a 9-step framework for teaching conceptual modelling, which is centred on design capture over a two-phase modelling task. She later proves that active learning methods regarding conceptual modelling are effective [8]. Chen Worlds [11] is an active learning tool for drawing ER diagrams, where students read a description and then choose from the visual alternatives curated by a tutor. MonstER Park [13] is a gamified learning tool in which students "talk to little monsters" that help them build an ER diagram and provide feedback.

Novelty. We see the need for a system based on the *classical* notion of Entity–Relation diagrams [6] to help students apply conceptual modelling practices for database design. The goal is a system that, unlike Chen Worlds and MonstER Park, is not too concerned with the visual notation of the ER diagram (arrows, lines, etc.), but rather with the concepts being modelled. The system should be available online, avoiding the need to install specialised software. It should further detect and help users to resolve both syntactic and semantic errors. We describe ERDoc: a system that satisfies these criteria. The system consists of the ERDoc syntax and language, used to define the ER diagram, and ERDoc Playground, a Web interface for defining, validating and visualising ER diagrams.

3 ERDOC LANGUAGE

ERDoc is a markup language for defining classical ER diagrams [6], including entities with their attributes and keys; relationships with attributes, cardinality, and participation constraints; as well as other features such as weak entities, class hierarchies, and aggregations.

ERDoc was designed to be succinct, borrowing notation style from Java, Python, and dbdiagram [1]. Figure 1 presents a small ERDoc document that specifies two entities and a relationship between them. Each entity is written starting with the `entity` keyword, followed by the name of the entity; the names of the attributes of the `entity` then go between curly braces, one per line. Attributes that are part of the primary key of the entity use the `key` modifier next to the attribute's name. For relationships, the `relation` keyword is used, followed by the relationship name. The names of the participating relationships are listed in parenthesis, separated by commas. Each entity can optionally have maximum cardinality (`1` or `N`) and participation (`!`, interpreted as at-least-one) constraints. The default value is `N` (zero or more). Figure 2a shows an ER diagram corresponding to the ERDoc document of Figure 1. A more complete example, featuring relationships with attributes, weak entities, and class hierarchies, can be found in Figure 3.

The complete grammar of ERDoc is presented in Figure 4, and a detailed explanation of each feature can be found in the docs: https://erdoc.dcc.uchile.cl/docs. Using the grammar's syntax and semantics, an ERDoc parser and a linter are implemented to detect syntactic and semantic errors, respectively. Examples of semantic errors are entities without key attributes, duplicate entities, relationships, etc.,

```
entity Employee {
    e_id key
    name }
entity Department {
    d_number key
    d_name }
relation Works_for(Employee N, Department 1!)
```

Figure 1: ERDoc document declaring two entities and a relationship among them in the ERDoc language.

relationships involving undefined entities, weak entities without a weak relation, circular dependencies between weak entities, etc.

4 ERDOC PLAYGROUND

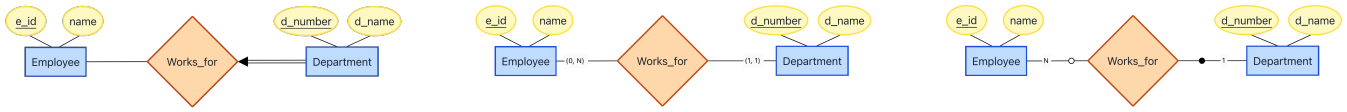
To help students learn and apply conceptual database modelling, and to facilitate the creation of ER diagrams, we have developed ERDoc Playground: a web application that leverages the ERDoc language to dynamically generate ER diagrams. The produced ER diagrams use Chen's original visual notation [6]. For displaying cardinality and participation constraints, ERDoc Playground supports three notations: Chen's, min/max, and arrow notation [12]. The ER diagrams corresponding to the ERDoc document in Figure 1 using the different notations are found in Figure 2. A demo of ERDoc Playground can be found at https://erdoc.dcc.uchile.cl.

4.1 User Interface

In Figure 5 we present the main UI of ERDoc Playground. On the left-hand side of the UI there is a text editor to write ERDoc documents. The application dynamically generates an ER diagram corresponding to the user input, which the user can run in two modes: with an automatic layout that positions elements for the user as they are added, or a manual layout that offers the user full control of positioning. The editor features syntax highlighting for the keywords in the ERDoc language. Both parser and linter enable users to get live semantic and syntactic error reporting on the ERDoc document as it is input. Below the text editor, there are two drawers. The top drawer lists all the errors in the ERDoc document, while the bottom drawer offers sample ERDoc documents to load.

On the right-hand side of the UI, the automatically generated ER diagram is displayed. Users can reposition the visual elements of the diagram (boxes, bubbles, etc.) by dragging them. Guiding lines are presented to better align new elements with existing elements both vertically and horizontally during manual layout. On the top-right of this panel, there is a button to open the configuration menu, where users can select the desired notation and the type of visualisation the edges of the diagram follow: straight lines or orthogonal paths (composed of vertical and horizontal segments).

The navigation bar at the top of the UI includes a switch to toggle the automatic layout of the diagram (see Section 4.2), buttons to load and save the state of the application (ERDoc document and ER diagram) as a JSON file, and buttons to export the diagram to image formats such as PDF, JPEG, and PNG. Here, users can also change the language of the application and visit the documentation.



(a) Arrow Notation. (b) Min/max notation. (c) Chen's notation.

Figure 2: ER diagram resulting from the document of Figure 1 with different notations.

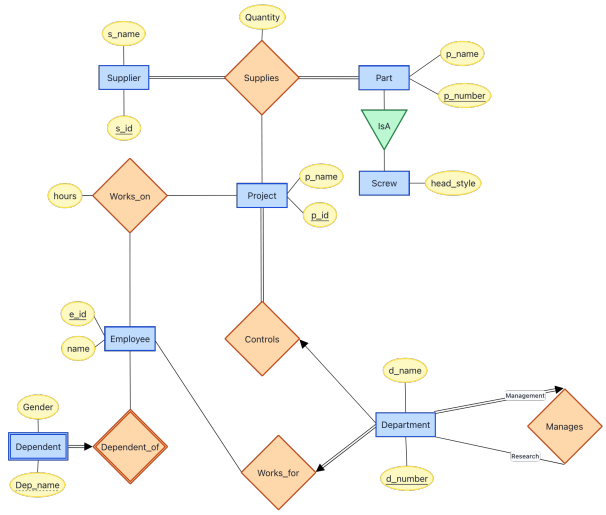


Figure 3: Large ER diagram with arrow notation.

The documentation (https://erdoc.dcc.uchile.cl/docs) contains sections about every ER diagram feature supported in ERDoc, where each section includes examples with their resulting ER diagrams.

4.2 Implementation

We now detail the implementation of ERDoc Playground, starting with the parsing and generation of the diagram, moving on to the methods used for automatically laying out the diagram, and finally summarising the architecture, languages, and environment used.

Parsing & diagram generation. To generate ER diagrams in real-time, the application parses the content of the text editor and checks for syntactic and semantic errors upon each user input. To encourage users to resolve these errors, the diagram is not generated until all syntactic and semantic errors are fixed. If there are no errors, the application then generates the corresponding ER diagram.

ER diagrams are internally represented as a graph containing a set of different types of nodes and edges (e.g., there is a specific node type to represent nodes, another for attributes, etc.). The process of generating the ER diagram is done by parsing the ERDoc input to its corresponding in-memory graph representation. This representation is then rendered considering the current selected notation. For example, the `Employee` entity in Figure 1 is internally represented as three nodes (one for the entity and one for each attribute) and two edges (connecting the entity with each attribute).

Automated layouts. Elements of the ER diagram, if placed in a default position, will overlap. On the other end, the user may not wish to concern themselves with manually placing elements, at least initially. Thus we offer an *Auto Layout* feature, which can be toggled on and off. When Auto Layout is on, a layout algorithm will be executed every time the diagram changes.

Adapting layout algorithms for ER diagrams in Chen notation is challenging because of the diverse ways in which the position of one node may depend on another. For example, when defining an aggregation, the relation and associated nodes are expected inside the boundary of the aggregation box; when defining a sub-class hierarchy, it is expected that the classes are presented as a tree; etc.

Our Auto Layout method first groups related elements (e.g., an entity and its attributes, a relationship and its entities, an aggregation and its relation, a sub-class hierarchy and its entities, etc.) into higher-level structures. For each such structure, we apply a different layout algorithm as we believe best suits the situation: *a*) To position the attributes of an entity or relationship, we use a *radial layout*, which renders neighbouring nodes surrounding a central node, balancing the edges' angles. *b*) We render class hierarchies in a tree-like visualisation using a *layer-based layout*, which positions the nodes in hierarchies, obtaining a visualisation where edges tend to point in the same direction. *c*) For the higher-level arrangement of entity, relationship, aggregation, and hierarchical structures, we use a *force-directed layout*, which simulates spring forces on the edges of the diagram, and repelling forces between the nodes, resulting in a visualisation with minimal overlaps between the structures. The user can also adjust element positions by hand.

Architecture, Languages & Environment. The application uses a typical client-server architecture. The front end runs on the client side and includes most of the application logic (parser, linter, visualiser, etc.). It uses TypeScript and React.js[2] to implement the user interface. The parser of ERDoc is written using PeggyJS[3], a parser generator for JavaScript. Graph layout is implemented on top of the ELK.js[4] library. A lightweight back end is implemented in Next.js[5] using TypeScript, and runs in a PM2[6] environment, which enables multiple instances to launch.

Demo & Source Code. An online demo of ERDoc Playground running on a Universidad de Chile server is available at https://erdoc.dcc.uchile.cl. The system is open source, with code available on GitHub at https://github.com/matias-lg/er.

[2] https://react.dev
[3] https://peggyjs.org/
[4] https://github.com/kieler/elkjs
[5] https://nextjs.org/
[6] https://pm2.keymetrics.io/

```
ERDocument := ERExpression MultilineDivider ERDocument | ERExpression
ERExpression := WeakEntityExpression | EntityExpression | RelationshipExpression | AggregationExpression
EntityExpression := 'ENTITY' EntityIdentifier ExtendsDecl? '{' EntityAttributeList '}'
ExtendsDecl := 'EXTENDS' EntityIdentifier
EntityAttributeList := EntityAttribute MultilineDivider EntityAttributeList | EntityAttribute
EntityAttribute := AttributeIdentifier CompositeDecl? 'key'?
CompositeDecl := ':' '[' AttributeIdentifierList ']'
AttributeIdentifierList := AttributeIdentifier ',' AttributeIdentifierList | AttributeIdentifier
WeakEntityExpression := 'ENTITY' EntityIdentifier WeakDecl '{' WeakEntityAttributeList '}'
WeakDecl := 'DEPENDS ON' DependenciesList
DependenciesList := RelationshipIdentifier ',' DependenciesList | RelationshipIdentifier
WeakEntityAttributeList := WeakEntityAttribute MultilineDivider WeakEntityAttributeList | WeakEntityAttribute
WeakEntityAttribute := AttributeIdentifier CompositeDecl? 'pkey'?
RelationshipExpression := 'RELATION' RelationshipIdentifier '(' ParticipatingEntitiesList ')' RelationshipAttributesDecl?
RelationshipAttributesDecl := '{' RelationshipAttributesList '}'
RelationshipAttributesList := AttributeIdentifier MultilineDivider RelationshipAttributesList | AttributeIdentifier
ParticipatingEntitiesList := ParticipatingEntity ',' ParticipatingEntitiesList | ParticipatingEntity
ParticipatingEntity := ParticipatingLabeledEntity | ParticipatingSingleEntity
ParticipatingLabeledEntity := EntityIdentifier ':' '[' ParticipatingSingleEntityList ']'
ParticipatingSingleEntityList := ParticipatingSingleEntity ',' ParticipatingSingleEntityList | ParticipatingSingleEntity
ParticipatingSingleEntity := EntityIdentifier EntityConstraintsDecl?
EntityConstraintsDecl := EntityCardinality '!'?
AggregationExpression := AggregationIdentifier '(' RelationshipIdentifier ')' EmptyCurlyBlock?
EmptyCurlyBlock := '{' '}'
EntityIdentifier, AttributeIdentifier, RelationshipIdentifier := id
```

Figure 4: Grammar of the ERDoc Language.

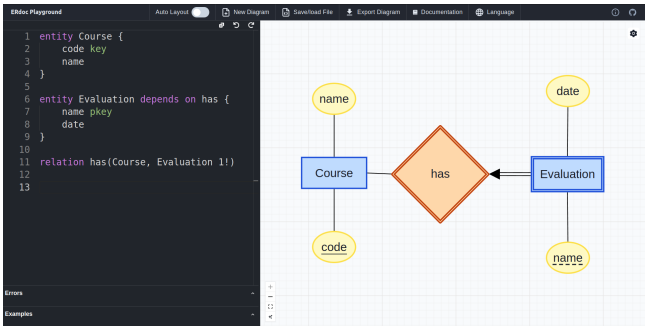


Figure 5: User Interface of ERDoc Playground.

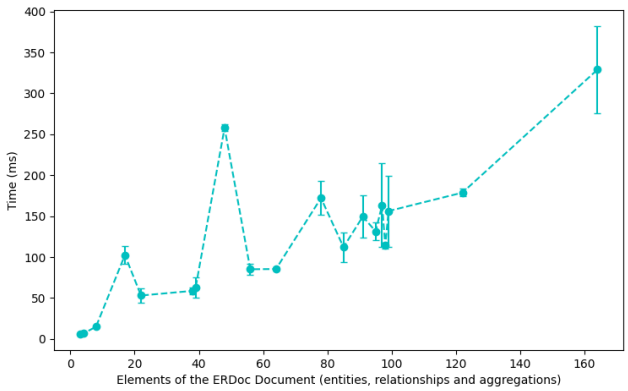


Figure 6: Execution time of Auto Layout.

5 EVALUATION

In this section, we present initial experiments we conducted to evaluate the performance and usability of ERDoc Playground.

5.1 Performance Evaluation

The goal of this part of the evaluation is to verify that ERDoc Playground can generate ER diagrams in real-time with little latency.

We run all the experiments in Node.js (version 18.17.1) on a machine with a 6-core AMD Ryzen™ 5 3600 CPU and 16 GB of RAM. The machine runs a 64-bit Ubuntu 22.04 OS. As input, we generate multiple ERDoc documents, each with a varying number of elements (entities, relationships, and aggregations).[7] The number of attributes for entities and relationships was randomly chosen (between 1 and 4). The generated ERDoc documents include class hierarchies as well. The largest ERDoc Document generated consists of 139 entities (including sub-classes), 20 relationships, and 5 aggregations; we estimate that, in the context of a learning tool, users may not need to create larger ER diagrams.

[7] Available at https://github.com/matias-lg/er/tree/perf-report/src/eval/reports.

For each ERDoc document, we measure the time it takes to execute the parser, linter, and graph generator as a whole. This process takes less than 6 *ms*, even for larger documents.

Then, for each ERDoc document, we measure the time taken by the Auto Layout (see Section 4.2) on the corresponding graphs generated by the previous step. Results in Figure 6 show that time increases with the number of elements in the document, and is usually below 400 *ms*. Note that the increase is not linear, as time also depends on the composition of the elements in the document.

Overall, the performance evaluation shows that the complete process of generating an ER diagram from user input takes less than 500 *ms* for documents with hundreds of elements. Thus, ERDoc Playground can generate small-to-moderate scale ER diagrams in near real-time, without evident latency, as users write their ERDoc document in the text editor. For larger diagrams, the layout algorithm would likely become a bottleneck, as the layout algorithms take time proportional to the number of edges in the graph. In this case, the user may choose to switch to manual layout.

5.2 User Evaluation

We conducted two usability surveys of ERDoc. The first survey was aimed at professors, teaching assistants and graduates from the Databases course. Users were asked (in Spanish) to use ERDoc to model a university database consisting of three entities, two relationships, and one weak entity. The requirements were phrased directly in ER terms, specifying the entities, relationships, etc., required. The survey indicated an approximate duration of 15 minutes, and subjects were encouraged to read ERDoc's documentation pages. The surveys were conducted online. The subjects had no previous experience with ERDoc. After the task was completed, we applied the System Usability Scale (SUS) [4] questionnaire, translated to Spanish by the authors. In addition to the SUS responses, we requested the ERDoc document produced by the subjects for the task and left a field for free-form comments. The form (in Spanish) we used can be found here: https://forms.gle/L8LRywdth5pJgxzi9. The second survey was aimed at current students of the Databases course, where ERDoc was used in the context of an ER lab. After the lab submission deadline, students were requested to fill out a SUS questionnaire, optionally leaving free-form comments.

The detailed SUS results for both surveys are presented in Table 1.

The first survey had 21 participants, of which 3 were previous course professors, 5 were teaching assistants, 3 were lab assistants, and 10 were students who had graduated from the Databases course. ERDoc Playground obtained an average SUS score of 83.45, with a standard deviation of 12.73. A SUS score higher than 68 is considered as *above average* [10]. Thus, the obtained score indicates a good perception of the usability of the application. Most subjects successfully completed the modelling task, submitting an ERDoc document that resembles the original database model. One subject submitted an empty ERDoc model. All of the non-empty ERDoc document submissions were syntactically correct and managed to correctly declare the model's entities and attributes, including key declarations and partial keys for weak entities. All the errors found in the submissions were related to cardinality and participation constraints in relationships, where subjects confused a one-to-many relationship with a many-to-one. There were nine completely correct ERDoc documents and eleven with errors in relationship constraints.

The second survey had 22 participants, all active students of the Databases course. Similarly to the initial survey, ERDoc Playground scored 84.40 (average) with a standard deviation of 14.70.

In the free-form comments, multiple users commend the responsiveness of the application and the real-time generation of the diagrams. Users mention that the syntax of ERDoc is intuitive and simple. They further requested a number of features, such as the ability to customise the colours, fonts, etc., used in the diagram. Students mentioned that the system was easy and intuitive to learn.

Table 1: SUS questionnaire scores. Scores range from 1 to 5. Lower scores are better for even-numbered items, whereas higher scores are better for odd-numbered items. m_1 and s_1 denote mean and standard deviation, respectively, for the initial evaluation of ERDoc Playground. Likewise, m_2 and s_2 denote the mean and standard deviation of the evaluation during the ER lab from the Databases course.

#	Question	m_1	s_1	m_2	s_2
1	I think I would like to use this system frequently.	4.10	0.83	4.23	0.81
2	I found the system unnecessarily complex	1.57	0.60	1.55	0.80
3	I thought the system was easy to use.	4.62	0.59	4.32	0.99
4	I think that I would need the support of a technical person to be able to use this system.	1.48	0.81	1.77	1.67
5	I found the various functions in this system were well integrated.	4.71	0.56	4.50	0.60
6	I thought there was too much inconsistency in this system.	1.57	0.81	1.32	0.72
7	I would imagine that most people would learn to use this system very quickly.	4.10	0.89	4.23	0.87
8	I found the system very cumbersome to use.	1.67	0.80	1.50	0.80
9	I felt very confident using the system.	4.24	0.89	4.45	0.86
10	I needed to learn a lot of things before I could get going with this system.	2.10	1.18	1.73	1.08

6 CONCLUDING REMARKS

We presented ERDoc: a syntax and language for describing ER diagrams, and an online system for helping users to design and generate ER diagrams. Based on the survey responses and the performance evaluation, we can conclude that ERDoc Playground is perceived as usable by users. With a low (sub-second) runtime of its main features (ER diagram generation, auto layout), it allows users to create small-to-moderate scale ER diagrams (with up to hundreds of elements) in real-time as they write in the text editor. We have further used ERDoc in laboratories on ER for our Databases course, with positive feedback from both students and teaching teams.

For future work, the graph layout module could be improved, creating more aesthetically-pleasing diagrams, though this would need to be balanced with the need for the layout to be generated in sub-second time. We could also offer better customisation of the look and feel of the application and the diagrams. We could provide support for mapping the ER specification to a relational model (though this is something we typically ask students to do). Finally, though the students, in general, found ERDoc to be useful, we could conduct more in-depth experiments to determine whether or not it has actually improved their conceptual modelling skills.

ACKNOWLEDGMENTS

This work was partly funded by ANID - Millennium Science Initiative Program - Code ICN17_002. López and Hogan were funded in part by FONDECYT Grant No. 1221926. We are also grateful to all those who participated in our usability studies.

REFERENCES

[1] [n. d.]. DBDiagrams. http://dbdiagram.io. [Online; accessed 26-April-2024].
[2] [n. d.]. MySQL Workbench. https://dev.mysql.com/doc/workbench/en/. [Online; accessed 26-April-2024].
[3] [n. d.]. pgModeler. https://pgmodeler.io/. [Online; accessed 26-April-2024].

[4] John Brooke. 1996. SUS: a "quick and dirty" usability scale. *Usability evaluation in industry* 189, 3 (1996), 189–194.
[5] Gonçalo Carvalho, Sergii Mykolyshyn, Bruno Cabral, Jorge Bernardino, and Vasco Pereira. 2022. Comparative Analysis of Data Modeling Design Tools. *IEEE Access* 10 (2022), 3351–3365. https://doi.org/10.1109/ACCESS.2021.3139071
[6] Peter Pin-Shan Chen. 1976. The entity-relationship model—toward a unified view of data. *ACM transactions on database systems (TODS)* 1, 1 (1976), 9–36.
[7] Karen C. Davis. 2014. Teaching Conceptual Design Capture. In *Advances in Conceptual Modeling*. Vol. 8697. Springer International Publishing, Cham, 247–256. https://doi.org/10.1007/978-3-319-14139-8_26
[8] Karen C. Davis. 2018. Teaching Physical Database Design. In *Advances in Conceptual Modeling*. Vol. 11158. Springer International Publishing, Cham, 165–175. https://doi.org/10.1007/978-3-030-01391-2_22
[9] Kamal Hingorani, Dexter Gittens, and Nicholas Edwards. 2017. Reinforcing Database Concepts By Using Entity Relationships Diagrams (ERD) and Normalization Together for Designing Robust Databases. *Issues in Information Systems* 18, 1 (2017), 148–155.
[10] Jeff Sauro. 2011. Measuring Usability with the System Usability Scale (SUS). https://measuringu.com/sus/
[11] Natalya Keberle and Ivan V Utkin. 2012. Teaching Conceptual Modeling in ER: Chen Worlds.. In *ICTERI*. 222–227.
[12] Raghu Ramakrishnan and Johannes. Gehrke. 2003. *Database Management Systems*. McGraw-Hill Education. https://books.google.cl/books?id=JSVhe-WLGZ0C
[13] Johannes Schildgen. 2020. MonstER Park-The Entity-Relationship-Diagram Learning Game.. In *ER Forum/Posters/Demos*. 150–157.
[14] John Miles Smith and Diane C. P. Smith. 1977. Database abstractions: aggregation and generalization. *ACM Trans. Database Syst.* 2, 2 (jun 1977), 105–133. https://doi.org/10.1145/320544.320546
[15] Li Yang and Li Cao. 2016. The effect of MySQL Workbench in teaching entity-relationship diagram (ERD) to relational schema mapping. *International Journal of Modern Education and Computer Science* 8, 7 (2016), 1.