Tablr: A Desktop Database Application

Software-ontwerp
2018-2019

Iteration 3

Contents

1 Introduction **2**

2 General Information **2**

2.1 Team Work 2
2.2 Iterations 3
2.3 The Software 3
2.4 Testing 3
2.5 UML Tools 4
2.6 What You Should Hand In 4
2.6.1 Late Submission Policy 5
2.6.2 When Toledo Fails 5
2.7 Evaluation 5
2.7.1 Presentation Of The Current Iteration 5
2.8 Peer/Self-assessment 6
2.9 Deadlines 6

3 Tablr **7**

4 Use Cases **10**

4.1 Use Case: Create Table 11
4.2 Use Case: Edit Table Name 11
4.3 Use Case: Edit Table Query 11
4.4 Use Case: Delete Table 13
4.5 Use Case: Open Table 13
4.6 Use Case: Add Column 13
4.7 Use Case: Edit Column Characteristic 13
4.8 Use Case: Delete Column 14
4.9 Use Case: Add Row 15
4.10 Use Case: Edit Row Value 15
4.11 Use Case: Delete Row 15
4.12 Use Case: Undo Modification 15
4.13 Use Case: Redo Modification 16

5 Implementation **16**

1 Introduction

For the course *Software-ontwerp*, you will design and develop *Tablr*, a desktop database application. The main challenge will be the user interface layer, which you will design from scratch. In Section 2, we explain how the project is organized, discuss the quality requirements for the software you will design and develop, and describe how we evaluate the solutions. In Section 3, we show a diagram of the domain model and explain the problem domain of the application. The use cases are described in detail in Section 4. Finally, we specify some implementation constraints in Section 5.

2 General Information

In this section, we explain how the project is organized, what is expected of the software you will develop and the deliverables you will hand in.

2.1 Team Work

For this project, you will work in groups of four. Each group is assigned an advisor from the educational staff. If you have any questions regarding the project, you can contact your advisor and schedule a meeting. When you come to the meeting, you are expected to prepare specific questions and have sufficient design documentation available. *If the design documentation is not of sufficient quality, the corresponding question will not be answered.* It is your own responsibility to organize meetings with your advisor and we advise to do this regularly. Experience from previous years shows that groups that regularly meet with their advisors produce a higher quality design. If there are problems within the group, you should immediately notify your advisor. Do not wait until right before the deadline or the exam!

To ensure that every team member practices all topics of the course, a number of roles are assigned by the team itself to the different members at the start of each iteration (or shortly thereafter in case of the first iteration). A team member that is assigned a certain role will give the presentation or demo corresponding to that role at the end of the iteration. That team member is *not supposed to do all of the work concerning his task*! He must, however, take a coordinating role in that activity (dividing the work, sending reminders about tasks to be done, make sure everything comes together, etc.), and be able to answer most questions on that topic during the evaluation. The following roles will be assigned round-robin:

Design Coordinator The design coordinator coordinates making the design of your software.

Testing Coordinator The testing coordinator coordinates the planning, designing, and writing of the tests for the software.

Domain Coordinator The domain coordinator coordinates the maintenance of the domain model.

As already mentioned, the goal of these roles is to make every team member participate in all aspects of the development of your system. **During each presentation or demo, every team member must be able to explain the used domain model, the design of the system, and the functioning of your test suite.**

2.2 Iterations

The project is divided into 3 iterations. In the first iteration, you will implement the base functionality of the software. In subsequent iterations, new functionality will be added and/or existing functionality will be changed.

2.3 The Software

The focus of this course is on the *quality* (maintainability, extensibility, stability, readability,...) of the software you write. We expect you to use the development process and the techniques that are taught in this course. One of the most important concepts are the General Responsibility Assignment Software Principles (GRASP). These allow you to talk and reason about an object oriented design. **You should be able to explain all your design decisions in terms of GRASP**.

You are required to provide class and method documentation as taught in previous courses (e.g. the OGP course). When designing and implementing your system, you should use a *defensive programming style*. This means that the *client* of the public interface of a class cannot bring the objects of that class, or objects of connected classes, into an inconsistent state.

Unless explicitly stated in the assignment, you do not have to take into account persistent storage, security, multi-threading, and networking. If you have doubts about other non-functional concerns, please ask your advisor.

2.4 Testing

All functionality of the software should be tested. **For each use case, there should be a dedicated scenario test class.** For each use case flow, there should be at least one test method that tests the flow. Make sure you group your test code per step in the use case flow, indicating the step in comments (e.g. `// Step 4b`). Scenario tests should not only cover success scenarios, but also negative scenarios, i.e., whether illegal input is handled defensively and exceptions are thrown as documented. You determine to which extent you use unit testing. The testing coordinator briefly motivates the choice during the evaluation of the iteration.

Tests should have good coverage, i.e. a testing strategy that leaves large portions of a software system untested is of low value. Several tools exist to give a rough estimate of how much code is tested. One such tool is Eclemma[1]. If this tool reports that only 60% of your code is covered by tests, this indicates there may be a serious problem with (the execution of) your testing strategy. However, be careful when drawing conclusions from both reported high coverage

[1]http://www.eclemma.org

and reported low coverage (and understand why you should be careful). The testing coordinator is expected to use a coverage tool and briefly report the results during the evaluation of the iteration.

2.5 UML Tools

There are many tools available to create UML diagrams depicting your design. You are free to use any of these as long as it produces correct UML. One of these UML tools is Visual Paradigm. Instructions to run Visual Paradigm in the computer labs is described in the following file:

```
/localhost/packages/visual_paradigm/README.CS.KULEUVEN.BE
```

This file also contains the location of the license key that you can use on your own computer.

2.6 What You Should Hand In

Exactly *one* person of the team hands in a ZIP-archive via Toledo. The archive contains the items below and follows the structure defined below. **Make sure that you use the prescribed directory names.**

- directory `groupXX` (where XX is your group number (e.g. 01, 12, ...))
  - `doc`: a folder containing the Javadoc documentation of your entire system
  - `diagrams`: a folder containing UML diagrams that describe your system (at least one structural overview of your entire design, and sufficient detailed structural and behavioural diagrams to illustrate every use case)
  - `src`: a folder containing your source code
  - `system.jar`: an executable JAR file of your system
  - `design.pdf` (optional): a document that clarifies your design and the main decisions; this document can be a prose text or a slide deck, or any other form that you deem appropriate.

When including your source code into the archive, make sure to *not include files from your version control system.* If you use subversion, you can do this with the the `svn export` command, which omits unnecessary repository folders from the source tree. Make sure you choose relevant file names for your analysis and design diagrams (e.g. `SSDsomeOperation.png`). You do **not** have to include the project file of your UML tool, only the exported diagrams. We should be able to start your system by executing the JAR file with the following command: `java -jar system.jar`.

Needless to say, the general rule that anything submitted by a student or group of students must have been authored exclusively by that student or group of students, and that accepting help from third parties constitutes exam fraud, applies here.

2.6.1 Late Submission Policy

If the zip file is submitted N minutes late, with $0 \leq N \leq 240$, the score for all team members is scaled by $(240 - N)/240$ for that iteration. For example, if your solution is submitted 30 minutes late, the score is scaled by 87.5%. So the maximum score for an iteration for which you can earn 4 points is reduced to 3.5. If the zip file is submitted more than 4 hours late, the score for all team members is 0 for that iteration.

2.6.2 When Toledo Fails

If the Toledo website is down – and *only* if Toledo is down – at the time of the deadline, submit your solution by e-mailing the ZIP-archive to your advisor. The timestamp of the departmental e-mail server counts as your submission time.

2.7 Evaluation

After iteration 1, and again after iteration 2, there will be an intermediate evaluation of your solution. An intermediate evaluation lasts 15 minutes and consists of: a presentation about the design and the testing approach, accompanied by a demo of the tests.

The intermediate evaluation of an iteration will cover only the part of the software that was developed during that iteration. Before the final exam, the *entire* project will be evaluated. It is your own responsibility to process the feedback, and discuss the results with your advisor.

The evaluation of an iteration is planned in the week after that iteration. Immediately after the evaluation is done, you mail the PDF file of your presentation to Prof. Bart Jacobs & Tom Holvoet and to your advisor.

2.7.1 Presentation Of The Current Iteration

The main part of the presentation should cover the design. The motivation of your design decisions *must* be explained in terms of GRASP principles. Use the appropriate design diagrams to illustrate how the most important parts of your software work. Your presentation should cover the following elements. Note that these are not necessarily all separate sections in the presentation.

1. An updated version of the domain model that includes the added concepts and associations.

2. A discussion of the high level design of the software (use GRASP patterns). Give a rationale for all the important design decisions your team has made.

3. A more detailed discussion of the parts that you think are the most interesting in terms of design (use GRASP patterns). Again we expect a rationale here for the important design decisions.

4. A discussion of the extensibility of the system. Briefly discuss how your system can deal with a number of change scenarios (e.g. extra constraints, additional domain concepts,...).

5. A discussion of the testing approach used in the current iteration.

6. An overview of the project management. Give an approximation of how many hours each team member worked. Use the following categories: group work, individual work, and study (excluding the classes and exercise sessions). In addition, insert a slide that describes the roles of the team members of the current iteration, and the roles for the next iteration. Note that these slides do not have to be presented, but we need the information.

Your presentation should not consist of slides filled with text, but of slides with clear design diagrams and keywords or a few short sentences. The goal of giving a presentation is to communicate a message, not to write a novel. All design diagrams should be *clearly readable* and use the correct UML notation. It is therefore typically a bad idea to create a single class diagram with all information. Instead, you can for example use an overview class diagram with only the most important classes, and use more detailed class diagrams to document specific parts of the system. Similarly, use appropriate interaction diagrams to illustrate the working of the most important (or complex) parts of the system.

2.8 Peer/Self-assessment

In order for you to critically reflect upon the contribution of each team member, you are asked to perform a peer/self-assessment within your team. For each team member (including yourself) and for each of the criteria below, you must give a score on the following scale: *poor/lacking/adequate/good/excellent*. The criteria to be used are:

- Design skills (use of GRASP and DESIGN patterns, ...)
- Coding skills (correctness, defensive programming, documentation,...)
- Testing skills (approach, test suite, coverage, ...)
- Collaboration (teamwork, communication, commitment)

In addition to the scores themselves, we expect you to briefly explain for each of the criteria why you have given these particular scores to each of the team members. The total length of your evaluation should not exceed 1 page.

Please be fair and to the point. Your team members will not have access to your evaluation report. If the reports reveal significant problems, the project advisor may discuss these issues with you and/or your team. Please note that your score for this course will be based on the quality of the work that has been delivered, and not on how you are rated by your other team members.

Submit your peer/self-assessment by e-mail to both Prof. Bart Jacobs & Tom Holvoet and your project advisor, using the following subject: **[SWOP] peer-/self-assessment of group $groupnumber$ by $firstname$ $lastname$**.

2.9 Deadlines

- The deadline for handing in the ZIP-archive on Toledo is **24 May, 2019, 4pm**.

- The deadline for submitting your peer/self-assessment is **26 May, 2019, 6pm**, by e-mail to both your project advisor and Prof. Bart Jacobs & Tom Holvoet.

3 Tablr

The goal of the project is to develop a desktop database application.

In this iteration, the application is extended with *computed tables*, *Form subwindows*, and undo-redo functionality.

Teams of 3 students need not implement undo-redo functionality. Teams of 2 students need not implement computed tables. Teams of 1 student need not implement either computed tables or undo-redo functionality.

The application's user interface consists of a single top-level window, whose content area shows zero or more rectangular *subwindows* on a gray background. Each subwindow cosists of a border, a title bar, a Close button, a horizontal and vertical scrollbar, and a content area. The user can resize a subwindow by dragging its borders or its corners, and move it by dragging its title bar, as usual. The user can close a subwindow by clicking its Close button. If the height of the content being shown by a subwindow is not greater than the height of the subwindow's content area, its vertical scrollbar is shown as disabled. Otherwise, the vertical scrollbar shows a scroll button whose height is such that the ratio of the height of the scroll button to the height of the scrollbar equals the ratio of the height of the content area to the height of the content. The user can drag the scroll button or click the scrollbar above or below the scroll button to make another part of the content visible in the content area, as usual. The horizontal scrollbar functions analogously.

At any point, if there is at least one subwindow, one subwindow is the *active subwindow*. It is shown in front of all other subwindows and it receives keyboard input, and it is distinguished visually from the other subwindows. Interacting with any subwindow using the mouse makes that subwindow the active subwindow. If the active subwindow is closed, and any subwindows remain, of those subwindows the one that was active most recently becomes again the active subwindow.

There are three kinds of subwindows:

- A Tables subwindow shows the list of the names of the tables that the user has created so far, along with their associated SQL query, in a tabular view. If the query is blank, the table is a *stored table*; otherwise the table is a *computed table*. The user can create and delete tables, edit their name and query, and open a table. The latter creates a new Table Design subwindow (if the table is a stored table and has no columns) or Table Rows subwindow (otherwise). The user can press Ctrl+T at any time to create a new Tables subwindow. The user can create a new Form subwindow for the currently selected table by pressing Ctrl+F.

- A Table Design subwindow shows the list of the columns of a particular stored table, in a tabular view. For each column, the name, the type, whether blanks are allowed, and the default value are shown. Whether blanks are allowed is shown in the form of a checkbox. The type is String, Email, Boolean, or Integer. A column's default value is shown as a checkbox if the column's type is Boolean, and in textual form otherwise. If a column allows blanks and its type is Boolean and its default value is blank, this is shown as a grayed-out checkbox. The user can add and remove columns and edit their characteristics. The user can press Ctrl+Enter to create a new Table Rows subwindow for the current table.

- A Table Rows subwindow shows the list of the rows of a particular table, in a tabular view. For each row, the value of that row for each of the table's columns is shown. Values for columns of type Boolean are shown as checkboxes; other values are shown in textual form. If a row's value for some column is blank and the column's type is Boolean, this is shown as a grayed-out checkbox. The user can add and remove rows (if the table is a stored table) and edit their values for the table's editable columns. The user can press Ctrl+Enter to create a new Table Design subwindow for the current table (if it is a stored table).

- A Form subwindow shows the values for a particular row of a particular (stored or computed) table, in a form-like view. The subwindow shows one *field* for each of the table's columns. The fields are arranged vertically, on a gray background, with some padding between the fields. The fields themselves show their value on a white background. To the left of each field, the name of the corresponding column is shown. At any time, the subwindow has a *current row number* k. Each field shows the value for the corresponding column of the k'th row of the underlying table. If the column is editable, the value can be edited through the field. The user can increase or decrease the current row number by pressing PageDown or PageUp, respectively. If the current row number is greater than the number of rows of the table, no fields are shown. If the underlying table is a stored table, a new row can be added by pressing Ctrl+N and the row corresponding to the current row number (if any) can be deleted by pressing Ctrl+D. A Form subwindow's title bar shows the name of the underlying table and the current row number.

The application shall support at least the following syntax for queries:

```
       Query  ::=  SELECT ColumnSpecs FROM TableSpecs WHERE Expr
 ColumnSpecs  ::=  ColumnSpec | ColumnSpec , ColumnSpecs
  ColumnSpec  ::=  Expr AS ColumnName
  TableSpecs  ::=  TableName AS RowId
                   | TableSpecs INNER JOIN TableName AS RowId
                     ON CellId = CellId
        Expr  ::=  TRUE | FALSE | LiteralNumber | LiteralString
                   | CellId | Expr Operator Expr | ( Expr )
    Operator  ::=  OR | AND | = | < | > | + | -
      CellId  ::=  RowId . ColumnName
```

Examples:

```
SELECT movie.title AS title FROM movies AS movie
```

```
    WHERE movie.imdb_score > 7
SELECT student.name AS name, student.program AS program
    FROM enrollments AS enrollment INNER JOIN students AS student
    ON enrollment.student_id = student.student_id
    WHERE enrollment.course_id = "SWOP"
SELECT parent.name AS parentName, child.name AS childName
    FROM persons AS parent
    INNER JOIN is_child_of AS link ON parent.id = link.parent_id
    INNER JOIN persons AS child ON link.child_id = child.id
    WHERE TRUE
```

If a query mentions a cell identifier $r.c$ and there is no column named c in the table corresponding to row identifier r, the query is *invalid*. The application may impose additional validity conditions as well. Trying to enter an invalid query behaves the same way as trying to enter an invalid value for a table cell.

Changing the name of a table or column is not allowed if any query refers to that table or column. (Note: This means that a modification of the query of a computed table is invalid if some other query refers to a column c of that table and after the modification the query no longer defines a column with name c.)

A column of a computed table is editable if its expression is invertible. An expression is invertible if it is a cell identifier that refers to an editable column, or it is a sum or difference of which one operand is a literal value and the other operand is an invertible expression. Editing the value for an editable cell of a computed table causes a corresponding change in the value of the cell of the underlying table from which the value of the editable cell was computed.

A subwindow's title bar shows the kind of subwindow, as well as, for a Table Design or Table Rows subwindow, the name of the table.

Any changes made through one subwindow are immediately visible in all other subwindows showing the same or derived information. However, invalid data entered in one subwindow do not propagate to other subwindows. Furthermore, if, for some piece of data, an invalid value was entered in one subwindow, and subsequently a valid value is entered in another subwindow, the valid value propagates to all subwindows showing the same information, including the ones showing an invalid value; the valid value overwrites any invalid values and any such invalid values are lost. This applies to computed cells as well: if the value of a cell in a stored table changes, the new value is propagated to all subwindows where that cell or any computed cell computed directly or indirectly from that cell is shown or being edited.

In any subwindow, the width of any of the columns shown in the tabular view can be modified by dragging the column header's right-hand border. Any changes made in this way to the layout of the tabular view are immediately visible in all subwindows showing the same information; for example, if the width of a column is changed in a Table Rows subwindow for a particular table, that column will immediately have the new width in any current or future Table Rows subwindow for that table. In other words, a separate tabular view layout is associated with each table's Table Rows view, as well as with each table's Table Design view, as well as with the Tables view.

A tabular view's logical height includes some space below the table where the

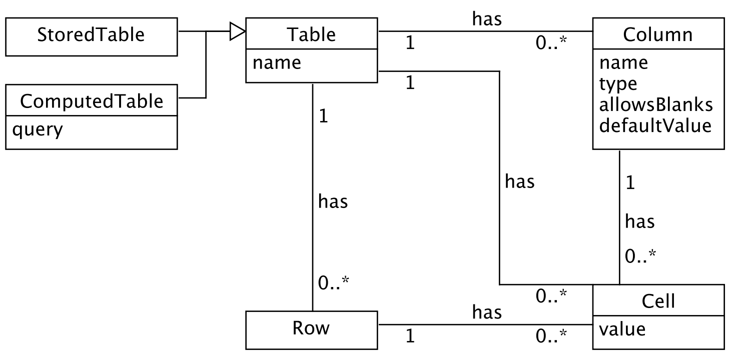

Figure 1: Domain model

user can double-click to add new rows.

Implement typical undo-redo functionality. Specifically, any user action that changes the database can be undone by pressing Ctrl+Z. Multiple consecutive user actions can be undone by pressing Ctrl+Z repeatedly. Recent undo operations can themselves be undone by pressing Shift+Ctrl+Z repeatedly, provided that the user performed no new original actions after the undo operations.

Changes to the state of the user interface caused by user actions or undo/redo operations are not undone by undoing the operation. For example: suppose the user creates a table, then opens a Table Design subwindow for that table, and then presses Ctrl+Z. The undo operation undoes the table creation and closes the Table Design subwindow. If the user then presses Shift+Ctrl+Z, the table creation is redone but the Table Design subwindow is not reopened.

Figure 1 shows the domain model of Tablr.

4 Use Cases

Figure 2 shows the use case diagram for Tablr. The following sections describe the various use cases in detail.

Notes:

- A system's requirements can be specified at various levels of abstraction with respect to the nature of the interface between the system and its environment (e.g. a human user). Often the specification abstracts over the nature of the interface to focus on the aspects that are most relevant to the usefulness of the system. However, in this assignment, since the main challenge concerns the design of the code that implements the user interface, we specify the interaction with the user in unusually specific detail.

- While the tool you develop should be functional, the user interface need not be of the level of "finish" that would be expected of a commercial product. For example:

- The text cursor need not blink.
- You need not (in this iteration) provide a menu, Save/Open functionality, printing functionality, etc.

4.1 Use Case: Create Table

Main Success Scenario:

1. The user double-clicks below the list of tables in a Tables subwindow.
2. A new table is added to the list. Its name is `Table`N, where N is such that the name is different from the names of the existing tables, and its query is blank. The new table initially has no columns and no rows.

4.2 Use Case: Edit Table Name

Main Success Scenario:

1. The user clicks a table name in a Tables subwindow.
2. The system shows a cursor after the table name.
3. The user edits the name by pressing Backspace to remove characters from the end of the table name, and character keys to add characters to the end of the table name.
4. The system shows the updated table name as it is being edited. If the current name is empty or equal to the name of another table, the system shows a red border around the table name to indicate that it is invalid.
5. The user presses Enter or clicks outside the table name in the content area of the same subwindow to finish editing the table name.
6. The system shows the new table name.

Extensions:

5a. The user presses Escape to cancel editing the table name.
   1. The table name is changed back to the value that was active when the user clicked the table name.

6a. The current table name is invalid or the table whose name is being edited is referred to by some query.
   1. The Enter key press or mouse click has no effect and the user can continue editing the table name.

4.3 Use Case: Edit Table Query

Main Success Scenario:

1. The user clicks a table's query in a Tables subwindow.
2. The system allows the query to be edited in the same way that it allows a table name to be edited. It also deals with invalid values in the same way.

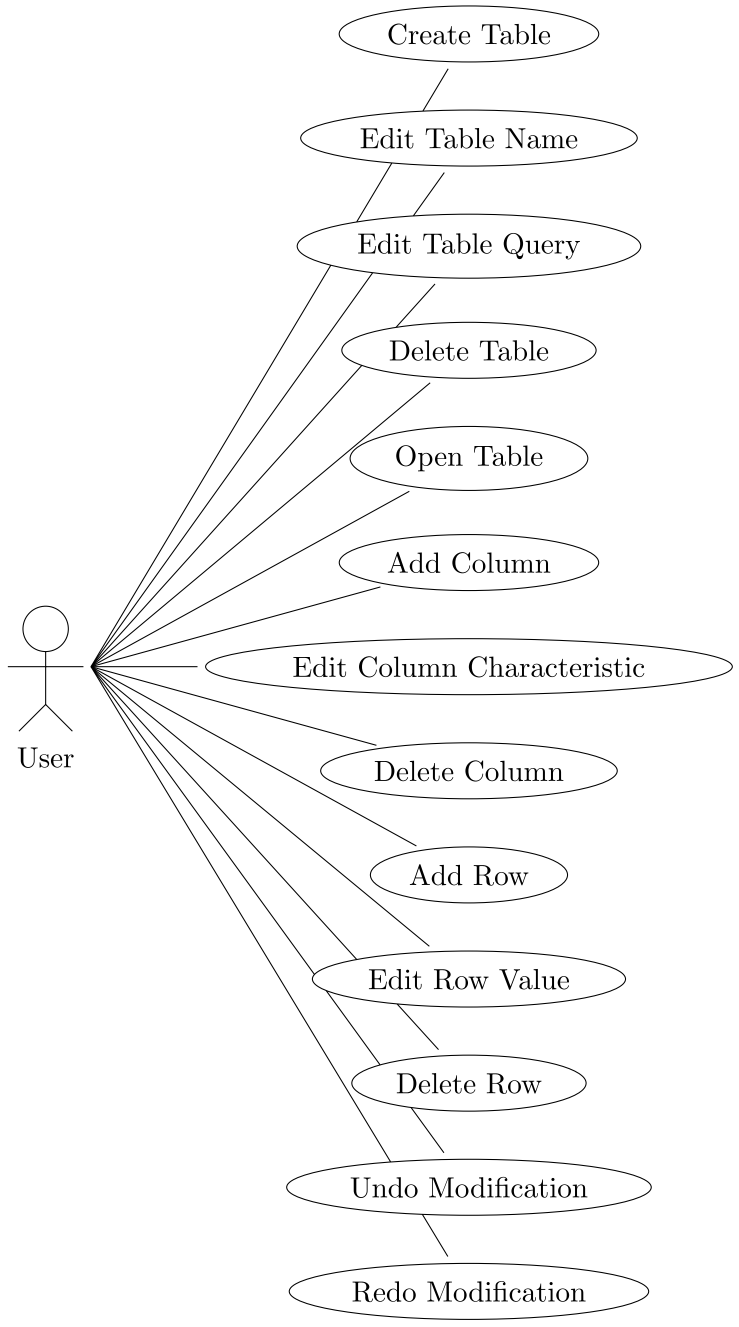

Figure 2: Use case diagram for Tablr.

4.4 Use Case: Delete Table

Main Success Scenario:

1. The user clicks the margin to the left of a table name in a Tables subwindow.
2. The system indicates that the table is now selected. Note: at any one point in time, different tables may be selected in different Tables subwindows.
3. The user presses the Delete key.
4. The system removes the table as well as all computed tables that directly or indirectly refer to it and shows the updated list of tables.

4.5 Use Case: Open Table

Main Success Scenario:

1. The user double-clicks a table name or presses Ctrl+F while a table is selected in a Tables subwindow.
2. The system creates a new Table Design subwindow for the double-clicked table (if the table is a stored table and has no columns), or a new Table Rows subwindow for the double-clicked table (otherwise), or a new Form subwindow for the selected table (if Ctrl+F was pressed).

4.6 Use Case: Add Column

Main Success Scenario:

1. The user double-clicks below the list of columns in a Table Design subwindow.
2. The system adds a new column to the end of the list. Its name is `Column`N, where N is such that the column name is different from the names of the existing columns; its type is String; blanks are allowed; and the default value is blank (i.e. the empty string)[2]. The system shows the updated list of columns. All existing rows of the table get a blank value as the value for the new column.

4.7 Use Case: Edit Column Characteristic

Main Success Scenario:

1. The user clicks the name of some column in a Table Design subwindow.
2. The system allows the user to edit the column name in the same way that it allows the user to edit a table name. A column name is valid if it is nonempty and different from the names of the other columns of the table unless some query refers to the column, in which case only the existing column name is valid.

[2]In contrast to many real database systems, there is no separate "blank" or null value that is different from the empty string

Extensions:

1a. The user clicks the type of some column.

1. If the type was String, it becomes Email. If it was Email, it becomes Boolean. If it was Boolean, it becomes Integer. If it was Integer, it becomes String. However, if the new type is such that the column's default value or the value of some row of the table for this column is now invalid, a red border is shown around the type and the user must click the type again before they can do anything else in the application.

1b. The user clicks the checkbox indicating whether blanks are allowed for some column.

1. If the checkbox was checked, it now becomes unchecked; otherwise, it becomes checked. However, if the column's default value was blank or the value of any of the table's rows for this column was blank and blanks are now no longer allowed, a red border is shown around the checkbox and the user must click the checkbox again before they can do anything else in the subwindow.

1c. The user clicks the default value for some column.

1. If the column's type is Boolean and the column allows blanks, then if the default value was True, it becomes False; if it was False, it becomes blank; and if it was blank, it becomes True. If the column's type is Boolean and the column does not allow blanks, then if the default value was True, it becomes False, and if it was False, it becomes True. If the column's type is not Boolean, then the system allows the user to edit the value in the same way that it allows the user to edit a table name. A default value is valid if either the default value is blank and the column allows blanks, or the column's type is String and the value is nonempty, or the column's type is Email and the default value contains exactly one @ character, or the column's type is Integer and the default value is the decimal representation (with no extraneous leading zeros) of an integer.

4.8 Use Case: Delete Column

Main Success Scenario:

1. The user clicks the margin to the left of some column's name in a Table Design subwindow.
2. The system indicates that the column is now selected.
3. The user presses the Delete key.
4. The system removes the column from the list of columns, and removes the value for the deleted column from all of the table's rows, and deletes all computed tables whose query directly or indirectly refers to the deleted column. This change is immediately visible in all subwindows that show the table's design or rows.

4.9 Use Case: Add Row

Main Success Scenario:

1. The user double-clicks below the list of rows in a Table Rows subwindow for a stored table, or presses Ctrl+N in a Form subwindow for a stored table.
2. The system adds a new row to the end of the table. Its value for each column is the column's default value.

4.10 Use Case: Edit Row Value

Main Success Scenario:

1. The user clicks the value of some row for some editable column in a Table Rows or Form subwindow.
2. The system updates the value or allows the user to edit the value, in the same way that it updates the default value or allows the user to edit the default value when the user clicks a column's default value in Table Design mode.

4.11 Use Case: Delete Row

Main Success Scenario:

1. The user clicks the margin to the left of some row in a Table Rows subwindow for a stored table.
2. The system indicates that this row is now selected.
3. The user presses the Delete key.
4. The system removes the row from the table and shows the updated list of rows.

Extensions:

1a. The user presses Ctrl+D in a Form subwindow for a stored table.

   1. The system removes the k'th row, where k is the Form subwindow's current row number, from the table and shows the values for the row that is now the k'th row, if any.

4.12 Use Case: Undo Modification

Precondition: The user has performed a database modification that they have redone as many times as they have undone it.

Main Success Scenario:

1. The user presses Ctrl+Z.
2. The system reverts the database to the state before the most recent modification that has been redone as many times as it has been undone.

4.13 Use Case: Redo Modification

Precondition: The user has performed a database modification that they have undone more often than they have redone it, and they have not performed another original modification since then.

Main Success Scenario:

1. The user presses Shift+Ctrl+Z.
2. The system reverts the database to the state after the most recent modification that has been undone more often than it has been redone.

5 Implementation

You must implement your system in a statically typed, object-oriented programming language.

Since the main intended challenge of this assignment is that you design your user interface layer from scratch, for this assignment you are not allowed to use an existing GUI toolkit, such as Java's Swing/AWT or SWT, or the .NET Framework's Windows Forms or Windows Presentation Framework. Instead, we require that you use only the CanvasWindow Java class that we provide, or an equivalent API that you implement yourself on another platform, to implement the user interface. (When using the provided CanvasWindow Java class, you can also use the AWT elements that are necessary to use this class, but you cannot use the AWT or Swing component hierarchies.)

Good luck!
The SWOP Team members