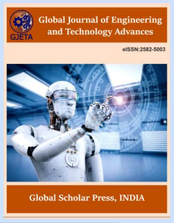
Global Journal of Engineering and Technology Advances

eISSN: 2582-5003

Cross Ref DOI: 10.30574/gjeta

Journal homepage: https://gjeta.com/



(Research Article)

On replacing standard system error messages for relational database constraints with context aware natural language ones

Diana Christina Mancas *

Department of Mathematics and Computer Science, Ovidius University at Constanta, Romania.

Global Journal of Engineering and Technology Advances, 2023, 14(03), 172–187

Publication history: Received on 18 February 2023; revised on 28 March 2023; accepted on 31 March 2023

Article DOI: https://doi.org/10.30574/gjeta.2023.14.3.0062

Abstract

The goal of this paper is to provide a rigorous methodology for enhancing VBA software application users' experience when faced with attempts to violate the underlying SQL Server database constraints. The original contribution is a pseudo-code algorithm for assisting developers in this process. We exemplify the results of using it with a full code example for a class taken from a genealogy database software application designed and developed using this methodology. Consequently, it replaces all context-independent and developer jargon standard system error messages with context aware and natural language ones. Moreover, it also synchronizes combo-box instances after each update or delete. We also show that the proposed algorithm works fine not only for other relational database management systems, but even with NoSQL platform backends, and that the modifications needed to adapt it to other similar SQL embedding frontend platforms are minimal. This algorithm main merit is that it may serve as the basis for developing a tool for performing this important task through automatic code generation.

Keywords: Relational database constraints; Event-driven programming; Database constraint-driven design and development; Automatic code generation; VBA; SQL Server

1. Introduction

Not only in the Database Constraint-Driven Design and Development (DCDDD) framework [1] of Software Engineering a final code polishing step in the software application development is included, for providing a user-friendly graphic user interface (GUI). Even if an application is correctly enforcing all business rules governing the corresponding sub-universe of discourse, almost nobody will accept at least to test it, not to speak of buying it, if its GUI is poorly designed and developed, no matter how glossy it looks. Generally, code polishing is done by developers on an ad-hoc basis: not even the best recent Software Engineering monographs (e.g., [2]) are proposing systematical approaches to it.

Database (db) designers take care at most of carefully designing db schemes, by including all needed columns in corresponding tables, as well as their (relational, i.e., provided by the Relational Database Management Systems (RDBMS)) constraints. These constraints are of the following six types (out of which the *default* type one may never be violated):

- *not-null* (i.e., corresponding table columns do not accept unknown values);
- *range* (i.e., corresponding columns accept values from only a subset of their data type);
- *referential integrity / foreign key* (i.e., corresponding columns accept only values stored in other table columns);
- *uniqueness / key* (i.e., corresponding columns do not accept duplicate values);
- *check / tuple* (i.e., values of several columns of a same table must satisfy a logic predicate for their values of any table data row), and

* Corresponding author: Diana Christina Mancas

- *default value* (i.e., when users do not provide a value for the corresponding column in the current data row, the system automatically assigns it the specified default one).

For example, in table *RULERS* from Fig. 10, there are:

- 3 not-null constraints (on columns *[#R]*, *Name*, and *Dynasty*);
- 4 range ones (on *[#R]*, *Name*, *BYear*, and *DYear*, where *autonum*. stands for system (unique) autonumbering and VARCHAR(64) stands for the set of all text strings of length at most 64);
- 2 referential integrity ones (on *Title*, which can take values only from those of column *x* of table *TITLES*, and on *Dynasty*, which can take values only from those of column *[#D]* of table *DYNASTIES*);
- 2 uniqueness ones (on *[#R]*, which is guaranteed by autonumbering, and on the pair <*Name*, *Dynasty*>, i.e., there may not be two persons of a same dynasty having same names), and
- 1 check / tuple one (*BYear* ≤ *DYear*, i.e., any person must be born at most in the same year as his/her death one).

Software application users might try sometimes to violate these constraints, either inadvertently or even on purpose, to test the application's quality. In such cases, backend RDBMSes reject corresponding invalid data values and return error and description codes to the frontend applications, which, for sparing additional programming effort, may choose to display them to their end-users. The issue in such cases is the fact that all system standard error messages are both context-independent and written in a developers' jargon.

The state-of-the-art solution to this issue is that some programmers are replacing at least some of these messages on an ad-hoc basis with better custom ones.

Consequently, we are introducing in this paper a pseudo-code algorithm for replacing all such system error ones with context-aware ones, preferably written in simple natural language. We chose as target technologies VBA (MS Office suite programming language [3]) over SQL Server [4] databases.

Obviously, such courtesy code methodologies heavily depend on the technological platforms used. However, we are also showing that our proposed one is easily adaptable to both other SQL embedding event-driven platforms than VBA (e.g., C#, Rust, Python, Java, etc.), as well as to other DBMS, be they relational (e.g., Oracle Database and MySQL, IBM DB2, Postgres, Access, etc.) or NoSQL (e.g., MongoDB, Cassandra, HBase, CouchDB, etc.).

The next section of this paper presents our proposed algorithm. Section 3 provides an example of applying it extracted from [5] and discusses contrastively these results vs. the lack of corresponding courtesy code. The paper ends with conclusion and further work, acknowledgement, and references.

2. Material and methods

Fig. 1 to 9 present our proposed algorithmic methodology (where ‘ starts comments).

3. Results and discussion

We applied in [5] our proposed algorithm to the development of a genealogy software application. Let us consider from its db only the 3 tables (with a few columns) shown in Fig. 10.

Fig. 11 shows the VBA code for class *TITLES* obtained by applying our proposed algorithm.

Here are the differences that this courtesy code brings to the application users' experience (where the even numbered messages are displayed by the code from Fig. 11, while the odd numbered ones are the corresponding system standard messages):

- When users inadvertently delete a title name, the messages are shown in Fig. 12 and 13; moreover, the application also restores the deleted value.
- When users ask for the deletion of a title that was associated with at least one person, the messages are shown in Fig. 14 and 15.
- When users ask for the deletion of a title that was not associated with anybody, the messages are shown in Fig. 16 and 17; moreover, if deletion is confirmed, the application re-queries the combo-box *Title* of the *RULERS*

form, which deletes that title from it as well, while the system does not this, but only replaces the name of the deleted title with the text “#Deleted”.

- When users try to duplicate an existing title, the messages are shown in Fig. 18 and 19.

```
Algorithm VBARelationalConstraintErrorMessagesHandling

Input: a set of MS Windows forms $\{F_1, \ldots, F_n\}$, $n > 0$, having as data sources the db tables
        $\{T_1, \ldots, T_m\}$, $0 < m \leq n$
Output: the set of forms $\{F_1, \ldots, F_n\}$ having their corresponding classes augmented with
        the VBA code needed for replacing context-independent system error messages
        when db table constraints would be violated with custom context-aware ones

Strategy:
For $i = 1$ To $n$
  Repeat for all constraint types $t$ of data source db table $T_j$ of form $F_i$
    Select Case t
      Case not-null NN(j, i)
      Case range RC(j, i)
      Case foreign key FK(j, i)
      Else UCC(j, i)                 ‘handle uniqueness and check/tuple type constraints
    End Select
  End Repeat;
  Del(m, i);                         ‘add code to prevent foreign key violation attempts
  Synchro(n, i);                     ‘add code to synchronize combo-box instances
  NoSQLInj(i);                       ‘add code to avoid basic SQL injection errors
Next i
End Algorithm VBARelationalConstraintErrorMessagesHandling
```

Figure 1 The proposed pseudo-code algorithm for assisting db constraint errors handling in VBA

```
NN(j As Integer, i As Integer)
If F_i does not have an associated Form_Error procedure Then CFEP(F_i);
add in its code immediately after the statement "Select Case DataErr" the following code:
        Case 3162      ‘SQL Server error code for attempt to violate a not-null constraint
          Select Case ctlCurrentControl.Name       ‘what is the current form control name?
          End Select
Repeat for all not-null columns c of table T_j
  If F_i does not have an associated c_BeforeUpdate procedure Then add it to its class;
  add to the beginning of the c_BeforeUpdate procedure the following VBA code:
        If Not Cancel And IsNull(c) Then              ‘handles c Is Null for new records
          Cancel = True                               ‘cancels emptying c
          Beep                                        ‘draws user attention
          MsgBox "Please specify a valid value for c.", vbCritical, "c is mandatory!"
          c.Undo                                      ‘restores previous c value
        End If
  add to the Form_Error procedure the following VBA code immediately after the
        "Select Case ctlCurrentControl.Name" statement following the "Case 3162" one:
          Case "c"                              ‘handles c Is Null for existing records
              MsgBox "Please specify a valid value for c.", vbCritical, "c is mandatory!"
              c.Undo                            ‘restores previous c value
End Repeat
```

Figure 2 The *NN* method called by the Algorithm *VBARelationalConstraintErrorMessagesHandling*

Please beware that SQL Server does not make any difference between either the possible several uniqueness keys of a table or its check constraint and the uniqueness keys.

```
RC(j As Integer, i As Integer)

If $F_i$ does not have an associated Form_Error procedure Then CFEP($F_i$);
add in its code immediately after the statement "Select Case DataErr" the following code:
        Case 3761, 7753, 2113        ‘SQL Server error codes for attempt to violate a range
                                     ‘constraint or enter not numeric values in numeric columns
           Select Case ctlCurrentControl.Name      ‘what is the current form control name?
           End Select
Repeat for all numeric columns c of table $T_j$ constrained by $c \in [minVal, maxVal]$
   If $F_i$ does not have an associated c_BeforeUpdate procedure Then add it to its class;
   add to the beginning of the c_BeforeUpdate procedure the following VBA code:
        If Not Cancel And ($c < minVal$ Or $c > maxVal$) Then    ‘handles c for new records
           Cancel = True                           ‘cancels modifying c
           Beep                                    ‘draws user attention
           MsgBox "Please specify a valid value for c.", vbCritical, _
                        "c must be between minVal  and maxVal!"
           c.Undo                                  ‘restores previous c value
        End If
   add to the Form_Error procedure the following VBA code immediately after the
        "Select Case ctlCurrentControl.Name" statement following the "Case 3761, …":
           Case "c"                       ‘handles c range for existing records
                MsgBox "Please specify a valid value for c.", vbCritical, _
                        "c values must be between minVal  and maxVal!"
                c.Undo                    ‘restores previous c value
End Repeat
```

Figure 3 The RC method called by the Algorithm *VBARelationalConstraintErrorMessagesHandling*

```
FK(j As Integer, i As Integer)
If Fi does not have an associated Form_Error procedure Then CFEP(Fi);
add in its code immediately after the statement "Select Case DataErr" the following code:
        Case 2237      ‘SQL Server error code for attempt to violate a foreign key constraint
           Select Case ctlCurrentControl.Name      ‘what is the current form control name?
           End Select
Repeat for all foreign key columns c of table Tj
   add to the Form_Error procedure the following VBA code immediately after the
        "Select Case ctlCurrentControl.Name" statement following the "Case 2237":
           Case "c"
                MsgBox "Please select a valid value for c.", vbCritical, _
                        "c must take values only from its combo-box!"
                c.Undo                          ‘restores previous c value
End Repeat
```

Figure 4 The *FK* method called by the Algorithm *VBARelationalConstraintErrorMessagesHandling*

```
CFEP(Fi As Form);
add a Form_Error procedure to the Fi ‘s class and add to its begining the following code:
        Dim ctlCurrentControl As Control           ‘declare a form control type variable
        Set ctlCurrentControl = Screen.ActiveControl      ‘store current control name
        Response = acDataErrContinue               ‘error handled
        Beep                                       ‘draws user attention
        Select Case DataErr                        ‘what is the SQL Server error code?
           Case Else                               ‘unexpected SQL Server error
                Response = acDataErrDisplay        ‘display system error message
        End Select
```

Figure 5 The *CFEP* method called by the methods from Fig. 2, 3, 4. and 6

```
UCC($j$ As Integer, $i$ As Integer)

If $F_i$ does not have an associated Form_Error procedure Then CFEP($F_i$);
add in its code immediately after the statement "Select Case DataErr" the following code,
for all $u$ unique keys $k_i = c_{ki1} \bullet \ldots \bullet c_{kiti}$ of table $T_j$, except for its autonumbering one $pkT_j$:
    Case 3146      'SQL Server error code for attempt to violate a unique or check constraint
        If Not IsNull(DLookup("*", " $T_j$,", "$c_{k11}$ =" & $c_{k11}$ & … & " And $c_{k1t1}$ =" & $c_{k1t1}$)) Then
            MsgBox "There is another element of $T_j$ having $c_{k11}$ =" & $c_{k11}$ & … & _
                    " And $c_{k1t1}$ =" & $c_{k1t1}$ & "!", vbCritical, "Request rejected…"
        ElseIf Not IsNull(DLookup("*", " $T_j$,", "$c_{k21}$ =" $c_{k21}$ & … & " And $c_{k2t2}$ =" & $c_{k2t2}$)) _
            Then MsgBox "There is another element of $T_j$ having $c_{k21}$ =" & $c_{k21}$ & … & _
                    " And $c_{k2t2}$ =" & $c_{k2t2}$ & "!", vbCritical, "Request rejected…"
        …
        ElseIf Not IsNull(DLookup("*", " $T_j$,", "$c_{ku1}$ =" $c_{ku1}$ & … & " And $c_{kutu}$ =" & $c_{kutu}$)) _
            Then MsgBox "There is another element of $T_j$ having $c_{ku1}$ =" & $c_{ku1}$ & … & _
                    " And $c_{kutu}$ =" & $c_{kutu}$ & "!", vbCritical, "Request rejected…"
        Else MsgBox "These values do not satisfy the check constraint of $T_j$!", vbCritical, _
                    "Request rejected…"
        End If
```

Figure 6 The UCC method called by the Algorithm *VBARelationalConstraintErrorMessagesHandling*

Please note that, moreover, our proposed algorithm also synchronizes combo-box instances after each update or delete in any table, through method *Synchro* (see Fig. 8).

To also avoid SQL injection errors, our proposed algorithm replaces apostrophes and quotes with their acute counterparts in any text box, through method *NoSQLInj* (see Fig. 9).

Both these add-on features contribute to application's user friendliness and bug-free coding.

```
Del(m As Integer, i As Integer)
If $F_i$ does not have an associated Form_BeforeDelConfirm procedure Then
   add it to its class and add to it the following statement, to prevent standard system
   deletion confirmation message displaying:
        Response = acDataErrContinue
If $F_i$ does not have an associated Form_Delete procedure Then add it to its class;
For j = 1 to m
  Repeat for all foreign key columns $fk_j$ of table $T_j$ referencing primary key $pkT_i$ of table $T_i$
     (the data source of form $F_i$) to add at the beginning of Form_Delete the following code:
      If Not Cancel And Not IsNull(DLookup("$pkT_j$", "$T_j$", " $fk_j$ =" & $pkT_i$)) Then
        Cancel = True              'cancels current $T_i$'s data row deletion
        Beep                       'draws user attention
        MsgBox "At least an element of table $T_j$ is referencing this element of $T_i$!", _
                    vbCritical, "Request rejected…"
      End If
  End Repeat
Next j
```

Figure 7 The *Del* method called by the Algorithm *VBARelationalConstraintErrorMessagesHandling*

Obviously, there are two main differences for the users between these two approaches:

- Without courtesy code, users get system messages, which are context-independent, while with courtesy code, they are context-dependent, hence much more user friendly.
- Courtesy code may and should use natural language in messages, while the system ones use computer programmers' jargon, which is sometimes not easy to understand even for computer programmers (e.g., those from Fig. 13, 15, and 19)!

```
Synchro(n As Integer, i As Integer)
add to $F_i$'s class the following custom procedure:
      Sub Synchronize()
      On Error Resume Next
      End Sub
For j = 1 to n
  Repeat for all foreign key columns $fk_j$ of table $T_j$ referencing primary key $pkT_i$ of table $T_i$
     (the data source of form $F_i$) adding before the "End Sub" statement the following code:
      Forms!$F_j$!$fk_j$.Requery
  End Repeat
Next j
If $F_i$ does not have associated Form_AfterUpdate and Form_AfterDelConfirm procedures
Then add them to its class;
add to the Form_AfterUpdate procedure as its first line the following statement:
        Synchronize
add to the Form_AfterDelConfirm procedure as its first line the following statement:
        If Status = acDeleteOK Then Synchronize
```

Figure 8 The *Synchro* method called by the Algorithm *VBARelationalConstraintErrorMessagesHandling*

```
NoSQLInj(i As Integer)
If there is no public ReplaceApostrophes method Then add the following code to a project module:
        Public Function ReplaceApostrophes(ByVal text As String) As String
        'returns text with apostrophes in it replaced as acute accent characters (´),
        'as well as quotes replaced with two acute accent characters (´´)
        Dim pos, l As Long
        pos = -1
```

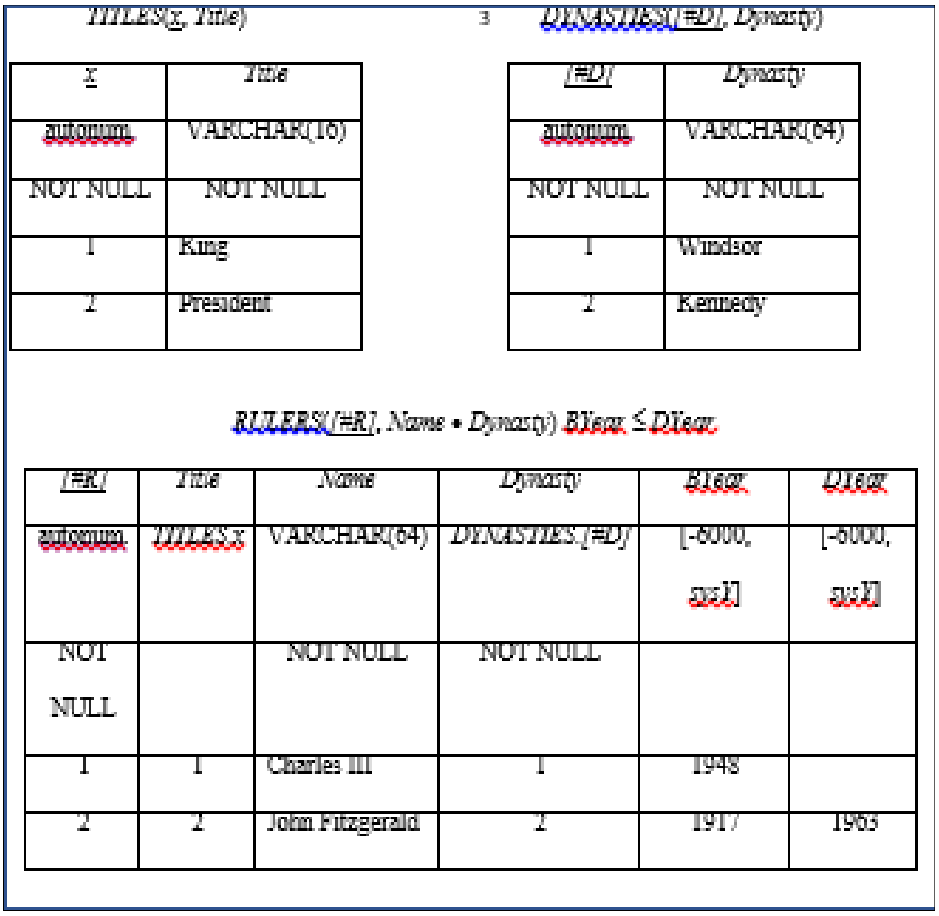
Figure 9 The *NoSQLInj* method called by the Algorithm *VBARelationalConstraintErrorMessagesHandli*

```
        l = Len(text)
        While pos <> 0
          pos = InStr(1, text, "'")
          If pos <> 0 Then
            text = Left(text, pos - 1) & "´" & Right(text, l - pos)
          End If
        Wend
        pos = -1
        While pos <> 0
          pos = InStr(1, text, """")
          If pos <> 0 Then
            text = Left(text, pos - 1) & "´´" & Right(text, l - pos)
          End If
        Wend
        ReplaceApostrophes = text
        End Function
Repeat for all text controls c of form $F_i$
    If $F_i$ does not have a c_AfterUpdate event procedure Then create it;
    add to the c_AfterUpdate event procedure as its first line the statement:
        c = ReplaceApostrophes(Trim(c))
End Repeat
```

Figure 9 (Continued)

Of course, that courtesy code needs additional programming effort, but, on one hand, it is clearly worthwhile and, on the other, this effort can be spared through automatic code generation [6, 7]. Obviously, automatic code generation cannot be done ad-hoc, but only algorithmically, which is the main strategic contribution of our proposed methodology.

The algorithm proposed in Section 2, which is the main contribution of this paper, can be easily adapted for similar programming platforms. For example, in MS .Net (e.g., C#) there exist similar to VBA event-driven procedures, only having different names (e.g., *BeforeUpdate* is called *Validating*, *AfterUpdate* is called *Validated*, etc.). Dually, thanks to the *de facto* standardization of both ADOX and ODBC, this algorithm may be used without modifications for the design of VBA software frontends based on other relational DBMSes



TITLES(x, Title) 3 *DYNASTIES([#D], Dynasty)*

x	*Title*
autonum	VARCHAR(16)
NOT NULL	NOT NULL
1	King
2	President

[#D]	*Dynasty*
autonum	VARCHAR(64)
NOT NULL	NOT NULL
1	Windsor
2	Kennedy

RULERS([#R], Name • Dynasty) BYear ≤ DYear

[#R]	*Title*	*Name*	*Dynasty*	*BYear*	*DYear*
autonum	*TITLES.x*	VARCHAR(64)	*DYNASTIES.[#D]*	[-6000, sysY]	[-6000, sysY]
NOT NULL		NOT NULL	NOT NULL		
1	1	Charles III	1	1948	
2	2	John Fitzgerald	2	1917	1963

Figure 10 A small genealogical db scheme

```
Private Sub Title_BeforeUpdate(Cancel As Integer)
If IsNull(Title) Then                    'enforces Title mandatory for new records
  Cancel = True
  Beep
  MsgBox "Please specify a title name.", vbCritical, "Title name is mandatory!"
  Title.Undo
End If
End Sub
Private Sub Title_AfterUpdate()
Title = ReplaceApostrophes(Trim(Title))          'prevents SQL injection errors
End Sub
Private Sub Form_AfterUpdate()
Synchronize
End Sub
Private Sub Form_Delete(Cancel As Integer)
Dim v, n As Variant
v = DLookup("[#R]", "RULERS", "Title =" & x)
If Not IsNull(v) Then                'at least one ruler had this title
  Cancel = True
  Beep
  n = DLookup("Name", "RULERS", "[#R] =" & CLng(v))
  v = DLookup("Dynasty", "RULERS", "[#R] =" & CLng(v))
  If IsNull(v) Then
    MsgBox "At least " & n & " (from an unknown dynasty) was a " & Title & "!", _
```

Figure 11 The courtesy code added to class *TITLES* for replacing system messages with custom ones

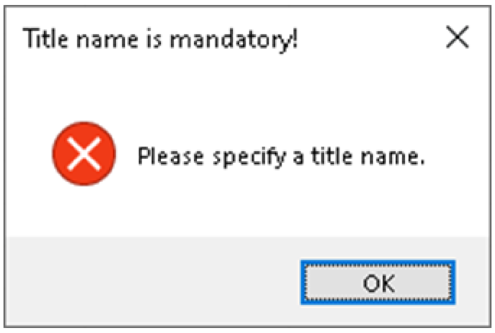
```
vbCritical, "Request rejected..."
  Else
    v = DLookup("Dynasty", "DYNASTIES", "[#D] =" & CLng(v))
    MsgBox "At least " & n & " from dynasty " & v & " was a " & Title & "!", _
            vbCritical, "Request rejected..."
  End If
End If
If Not Cancel Then     'asks corresponding confirmation message
  If vbCancel = MsgBox("Are you sure you want to delete the title " & Title & "?", _
        vbQuestion + vbOKCancel + vbDefaultButton2, "Please confirm or cancel...") _
  Then Cancel = True
End If
End Sub
Private Sub Form_BeforeDelConfirm(Cancel As Integer, Response As Integer)
Response = acDataErrContinue          'prevents system displaying its standard deletion message
End Sub
Private Sub Form_AfterDelConfirm(Status As Integer)
If Status = acDeleteOK Then Synchronize
End Sub
Private Sub Form_Error(DataErr As Integer, Response As Integer)
Response = acDataErrContinue          ' handles SQL Server errors
Beep
Select Case DataErr
  Case 3146                           ' SQL Server attempt to duplicate key values error code
```

Figure 11 (Continued)

e.g., Oracle's DB and MySQL, IBM's DB2, Postgres DB, MS Access, etc.): only minor changes in VBA coding are needed (e.g., correspondingly replacing constants storing DBMS error codes, taking care of particular behaviours, like the fact that autonumbering values are generated by MS Access immediately after users type a character on a new data line, which makes them available as early as the *Form_AfterInsert* event-driven procedure, while MS SQL Server does it only after saving the new data line, which makes them available only in the *Form_AfterUpdate* event procedure, etc.).

```
  MsgBox "The title " & Title & " is already stored: please change the title name!", _
      vbCritical, "There may not be two titles with a same name..."
  Title.Undo
Case 3162                              ' SQL Server attempt to violate not null value error code
  MsgBox "Please specify a title name.", vbCritical, "Title name is mandatory!"
  Title.Undo
Case Else   ' unexpected SQL Server error
  Response = acDataErrDisplay
End Select
End Sub
Sub Synchronize()
  On Error Resume Next
  Forms!RULERS!Title.Requery
End Sub
```

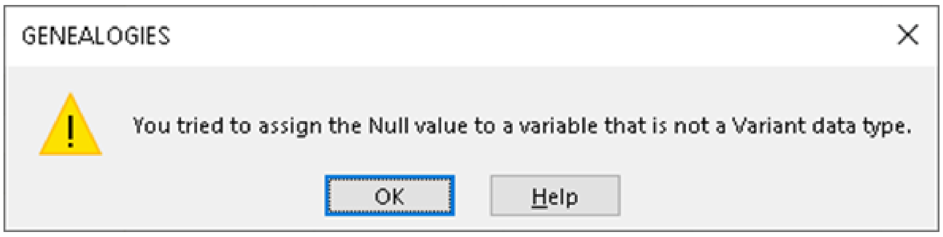
Figure 11 (Continued)



Title name is mandatory!

Please specify a title name.

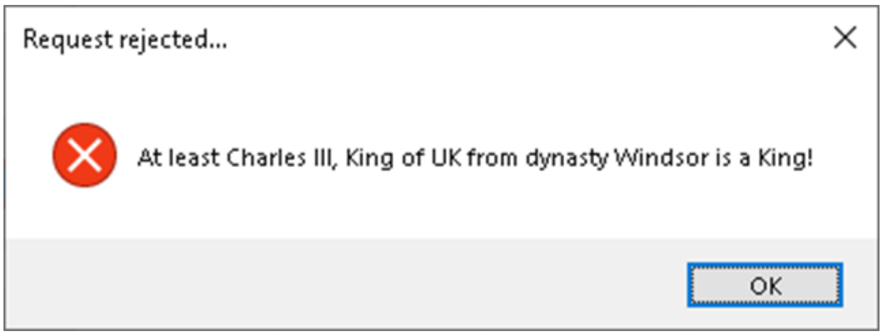
Figure 12 App. message when a title name is deleted



GENEALOGIES

You tried to assign the Null value to a variable that is not a Variant data type.

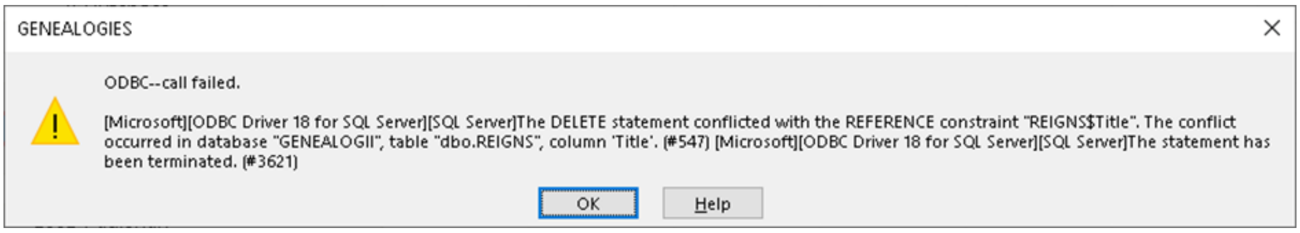
Figure 13 Standard system message when a title name is deleted



```
Request rejected...

At least Charles III, King of UK from dynasty Windsor is a King!
```

Figure 14 App. message when trying to delete the title “King”

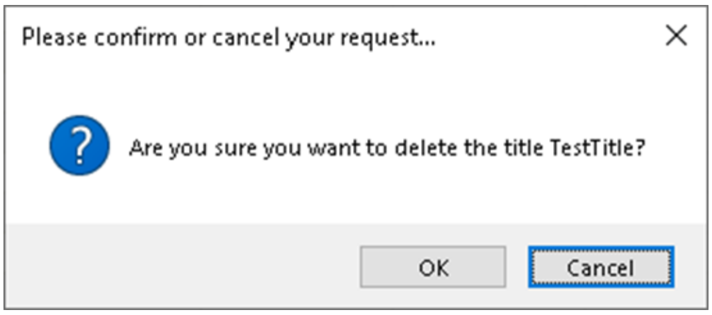


```
GENEALOGIES

ODBC--call failed.

[Microsoft][ODBC Driver 18 for SQL Server][SQL Server]The DELETE statement conflicted with the REFERENCE constraint "REIGNS$Title". The conflict
occurred in database "GENEALOGII", table "dbo.REIGNS", column 'Title'. (#547) [Microsoft][ODBC Driver 18 for SQL Server][SQL Server]The statement has
been terminated. (#3621)
```

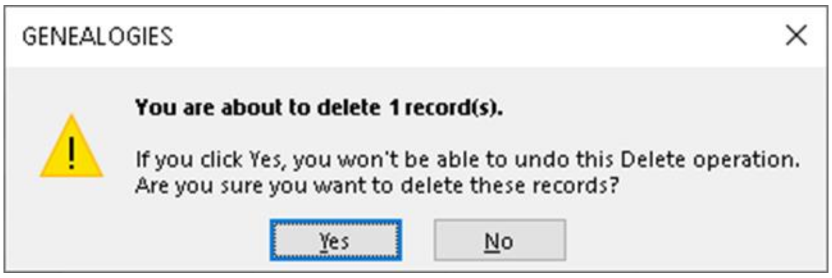
Figure 15 Standard system message when trying to delete the title “King”



```
Please confirm or cancel your request...

Are you sure you want to delete the title TestTitle?
```

Figure 16 App. message when trying to delete the title “TestTitle”

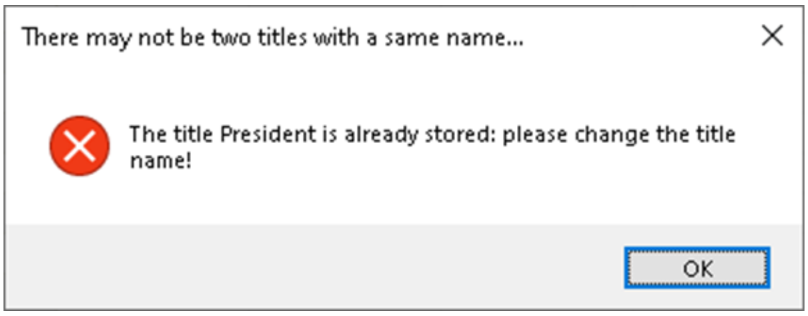


```
GENEALOGIES

You are about to delete 1 record(s).

If you click Yes, you won't be able to undo this Delete operation.
Are you sure you want to delete these records?
```

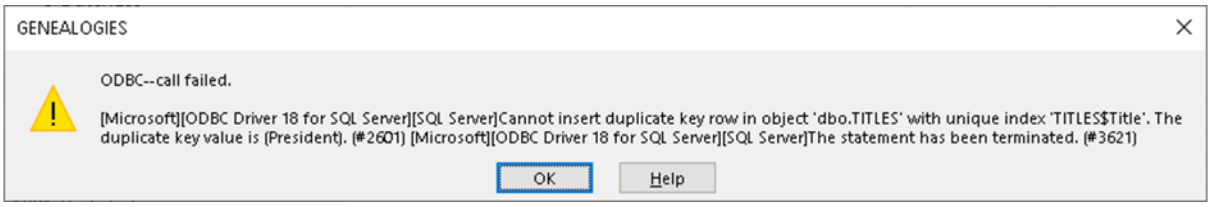
Figure 17 Standard system message when trying to delete the title “TestTitle”



```
There may not be two titles with a same name...

The title President is already stored: please change the title
name!
```

Figure 18 App. message when trying to duplicate title “President”



```
GENEALOGIES

ODBC--call failed.

[Microsoft][ODBC Driver 18 for SQL Server][SQL Server]Cannot insert duplicate key row in object 'dbo.TITLES' with unique index 'TITLES$Title'. The
duplicate key value is (President). (#2601) [Microsoft][ODBC Driver 18 for SQL Server][SQL Server]The statement has been terminated. (#3621)
```

Figure 19 System standard message when trying to duplicate title “President”

Moreover, our proposed algorithm could also be used for software applications built on top of NoSQL DBMSes: as they lack support for even almost all the relational constraint types (generally, except for one uniqueness constraint per table), the only difference is that the set of the relational constraints to be monitored by the frontend is much smaller, so our proposed algorithm becomes much simpler.

4. Conclusion and further work

In the framework of the Database Constraint-Driven Design and Development software engineering methodology, we designed, proposed, and used an algorithm for assisting the process of enhancing the frontend MS VBA classes of the Windows forms, aimed to replace standard system error messages for attempts to violate relational db constraints, which are both context-independent and written in developers' jargon, with context-aware ones, preferably written in everyday language. Moreover, this algorithm synchronizes corresponding combo-box instances after each update or delete and prevents basic SQL injection bugs.

Where needed and possible, we also provided actual corresponding VBA code patterns. We exemplified the result of applying this algorithm with the VBA code of a class taken from a genealogy db software application.

As usual, many developers might not use our algorithm, but, what is it much more important is that this algorithm may be used as a basis for developing a tool for corresponding automatic code generation, which is the future of computer programming.

Further work will be done, first to exemplify the power of our proposed algorithm even in the frameworks using other DBMSes, both relational and NoSQL; secondly, devising similar algorithms for other SQL embedding platforms; thirdly, developing a tool for automatic code generation based on these algorithms.

Compliance with ethical standards

Acknowledgments

The author is grateful to Associate Professor Cristina Serban, who is the scientific coordinator of her MSc. dissertation thesis.

References

[1] Mancas C, Serban C, Mancas DC. On Software Application Database Constraint-Driven Design and Development. Journal of Computer Science Research. 2023 Mar; 5(1):31-45, DOI: https://doi.org/10.30564/jcsr.v5i1.5476.

[2] Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. London, UK: O'Reilly; 2016. ISBN-13: 978-1449373320.

[3] Microsoft Corp. Office VBA Reference [Internet]. Redmond: Microsoft Corp.; © 2023 [cited 2023 Mar 25]. Available from Office Visual Basic for Applications (VBA) reference | Microsoft Learn.

[4] Microsoft Corp. Transact-SQL Reference (Database Engine) [Internet]. Redmond: Microsoft Corp.; © 2023 [cited 2023 Mar 25]. Available from Transact-SQL reference (Database Engine) - SQL Server | Microsoft Learn.

[5] Mancas DC. Design and Development of a Database Software Application for Managing Genealogical Trees [M.Sc. dissertation]. Constanta, Romania: Ovidius University; 2023.

[6] Thalheim B., Jaakkola H. Models as Programs: The Envisioned and Principal Key to True Fifth Generation Programming. In: Dahanayake A. and Huiskonen J., eds. Conference Proceedings of 29th International Conference on Information Modelling and Knowledge Bases - EJC 2019. Lappeenranta: IOS Press; 2019, pp. 147–166.

[7] Mancas C. On Modelware as the 5th Generation of Programming Languages. Acta Scientific Computer Sciences. 2020 Sep; 2(9):24–26.

Author's short biography

Diana Christina Mancas graduated in 2021 from the Faculty of Engineering Taught in Foreign Languages (Computers and Information Technology in French stream) of the Politehnica University at Bucharest, Romania.

From 2021 she is a MSc. student with the Mathematics and Computer Science Faculty of the Ovidius University at Constanta, Romania. She is currently preparing for her MSc. dissertation thesis defence, under the scientific supervision of Associate Professor Cristina Serban.