Teaching Tip

Specification and Enforcement of Semantic Integrity Constraints in Microsoft Access

Mohammad Dadashzadeh
Decision and Information Sciences Department
Oakland University
Rochester, Michigan 48309, USA

ABSTRACT

Semantic integrity constraints are business-specific rules that limit the permissible values in a database. For example, a university rule dictating that an *incomplete* grade cannot be changed to an *A* constrains the possible states of the database. To maintain database integrity, business rules should be identified in the course of database design and must be enforced during database implementation. Unfortunately, the manner by which current database management systems (DBMS) support the specification and enforcement of semantic integrity constraints varies considerably. This paper categorizes semantic integrity constraints and presents a simple approach for teaching users how to implement each of the five categories in Microsoft Access.

Keywords: Database Management Systems, Semantic Integrity Constraints, Validation Rules, SQL, Microsoft Access.

1. INTRODUCTION

Semantic integrity constraints, also referred to as data validation rules, are a common occurrence in database implementation scenarios. Consider the following database about departments and their employees:

DEPT(DNO, Name, City, ManagerID, PayrollBudget)
EMP(ENO, FullName, HireDate, ReviewDate, Salary, SSN, DeptID)

Each of the following represents a sample business rule or semantic integrity constraint for this scenario.

Rule 1: No two departments will be assigned the same value for DNO (primary key integrity).
Rule 2: No two employees will be assigned the same value for ENO (primary key integrity).
Rule 3: Social Security Number (SSN) is either null (missing) or is unique, that is, no two employees will have the same value for SSN.
Rule 4: ManagerID is either null or the same as an ENO value (referential integrity).
Rule 5: DeptID must be the same as a DNO value (referential integrity).
Rule 6: The only permissible values for City are: Boston, Chicago, and Detroit.
Rule 7: PayrollBudget for each department must be greater than or equal to the sum of salaries of employees assigned to that department.
Rule 8: HireDate must be greater than or equal September 1, 2005.
Rule 9: For each employee, ReviewDate is either null or greater than HireDate.
Rule 10: Valid salaries are between $10,000 and $90,000.
Rule 11: Department D10 employee salary cannot be less than $35,000.
Rule 12: Salaries should not be reduced.
Rule 13: For department D10 employees hired on the same date, the ReviewDate must be identical.
Rule 14: Each department manager must come from the same department.

To maintain our example database's integrity, each of the above rules must be enforced. There are four ways to accomplish this:

1. Let the **users** be responsible for it!
2. Do not let users update (i.e., add, delete, or modify) the database directly. Always, write programs to handle data entry and update, and let the **programs** enforce the integrity constraints.
3. Let the users update the database directly, but write DBMS ***triggers*** that would be invoked automatically upon updates to enforce the integrity constraints.
4. Declare the integrity constraints as DBMS ***assertions*** that would automatically be enforced by the DBMS.

It is generally agreed that letting the users police themselves would be an unrealistic approach. On the other hand, the most ideal approach is through DBMS assertions where the burden of enforcement is placed completely on the DBMS itself. Unfortunately, current database management systems fall short of this ideal (Türker and Gertz, 2001) and database developers must resort to some form of programming (approaches 2 and 3) to enforce integrity constraints.

The concept of database assertions is not new (Date, 1990; Grefen and Apers, 1993). Indeed, database assertions are supported in a limited basis by all current DBMS software. Specifically, the ability to designate the primary key of a table is nothing more than asserting a constraint and letting the DBMS enforce it during database updates. On the other hand, an integrity constraint such as Rule 7 could be specified in SQL-99 syntax as:

```
CREATE ASSERTION Rule7
   CHECK Not Exists
   (SELECT      DNO
   FROM         DEPT INNER JOIN EMP ON
                DEPT.DNO = EMP.DEPTID
   GROUP BY     DNO, PayrollBudget
   HAVING       Sum( EMP.Salary ) > PayrollBudget);
```

leaving its enforcement to the DBMS. Unfortunately, support for such arbitrary database assertions is not present in today's commercial software.

In the absence of support for database assertions, an integrity constraint such as Rule 7 can be enforced by programming similar to:

```
If Exists (
   SELECT       DNO
   FROM         DEPT INNER JOIN EMP ON
                DEPT.DNO = EMP.DEPTID
   GROUP BY     DNO, PayrollBudget
   HAVING       Sum( EMP.Salary ) > PayrollBudget )
Then
   Alert("Rule 7 has been violated!")
   Abort
End If
```

Of course, the preceding sample code needs to be executed whenever a new employee row is added or when the Salary or DeptID fields are changed. When a DBMS supports the concept of triggers, code such as the above can be written once and associated with the table EMP for automatic execution whenever certain events trigger (in this case, when a row is inserted, or when rows, or specifically, when Salary or DeptID are changed). Importantly, the code will be automatically triggered no matter how the update originates. That is, whether the user is explicitly executing an SQL UPDATE statement or a program supporting a user data entry/update form is making the update implicitly. If a DBMS does not support the concept of triggers, as is the case with Microsoft Access, then the code must be associated with *each* data entry/update form that could potentially insert a new row in the EMP table or modify the Salary and/or DeptID fields. Furthermore, to ensure that Rule 7 is not violated, the users should be prevented from explicitly issuing SQL INSERT and UPDATE statements against the EMP table.

As such, Microsoft Access provides mixed support for specification and enforcement of semantic integrity constraints. It supports database assertions in a limited way, does not support triggers, but provides the necessary methods for procedural support of enforcing integrity constraints. In this paper, we present a classification of semantic integrity constraints and give a simple approach for teaching users how to implement each of the five categories in Microsoft Access.

2. A CLASSIFICATION OF SEMANTIC INTEGRITY CONSTRAINTS

Semantic integrity constraints are either *static* or *dynamic*. Static integrity constraints are those constraints that can be determined to have been violated or not by a transaction simply by examining the database state when the transaction commits its changes. For example, consider a database transaction change of an employee's salary to:

EMP						
ENO	Full Name	Hire Date	Review Date	Salary	SSN	DEPT ID
E1	Emily Smith	9/1/2005		$86,000	123-45-6789	D10

Rules 10 and 11 can be immediately verified to have not been violated by merely considering this proposed database state. Specifically, salary is within permissible range of $10,000 to $90,000 and, as a department D10 employee, the salary is indeed not less than $35,000. However, it is impossible to verify that the transaction has not violated Rule 12 by merely examining the proposed database state. A dynamic integrity constraint, such as Rule 12, can only be verified by examining *both* the proposed database state as well as the starting database state. So, if the prior state is:

then Rule 12 is seen as being violated since the salary value of $87,500 is being reduced to $86,000.

Static integrity constraints can be classified into four categories:

Domain Type constraints limit permissible values for a data field (column). A domain type constraint can be determined to have been violated or not by simply examining the value of a

ENO	Full Name	Hire Date	Review Date	Salary	SSN	DEPT ID
E1	Emily Smith	9/1/2005		$87,500	123-45-6789	D10

Static integrity constraints can be classified into four categories:

Domain Type constraints limit permissible values for a data field (column). A domain type constraint can be determined to have been violated or not by simply examining the value of a single field in the record being added/changed. For example, valid salaries are between $10,000 and $90,000 (Rule 10).

Tuple Type constraints limit permissible values for a data field based on values in other data fields in the same record. A tuple type constraint can be determined to have been violated or not by examining the values of multiple fields in the record being added/changed. For example, department D10 employee salary cannot be less than $35,000 (Rule 11).

Relation Type constraints limit permissible values for a data field based on values in other records in the same table. A relation type constraint can be determined to have been violated or not by examining the values in other records in the table being added/changed. The quintessential relation type constraint is the primary key integrity rule that limits the permissible value in the key field of a record being inserted/changed by the existence of the same value in other records in the table.

Database Type constraints limit permissible values for a data field based on values in other records in other tables in the database. A database type constraint can be determined to have been violated or not by examining the values in other records in other tables. The quintessential database type constraint is the referential integrity rule that limits the permissible value in the foreign key field of a record being inserted/changed by the existence of the same value in the primary key field of another table in the database.

Table 1 classifies each of the 14 rules in our example database scenario and the next section presents the manner by which each type can be enforced in Microsoft Access.

3. SEMANTIC INTENGRITY CONSTRAINTS IN MICROSOFT ACCESS

Microsoft Access provides mixed support for specification and enforcement of semantic integrity constraints. Domain type constraints are handled easily through assertions as validation rules. Figure 1 shows the specification of Rule 6 in the table design view in Access.

Tuple type constraints in Microsoft Access are also handled by assertions. However, all tuple type constraints must be combined in a single validation rule specified as a table property. To assert Rules 9 and 10, the following combined validation rule must be specified as shown in Figure 2:

```
(([ReviewDate] Is Null) Or ([ReviewDate]>[HireDate])) And
(([DEPTID]<>"D10") Or (([DEPTID]="D10") And
([Salary]>35000))).
```

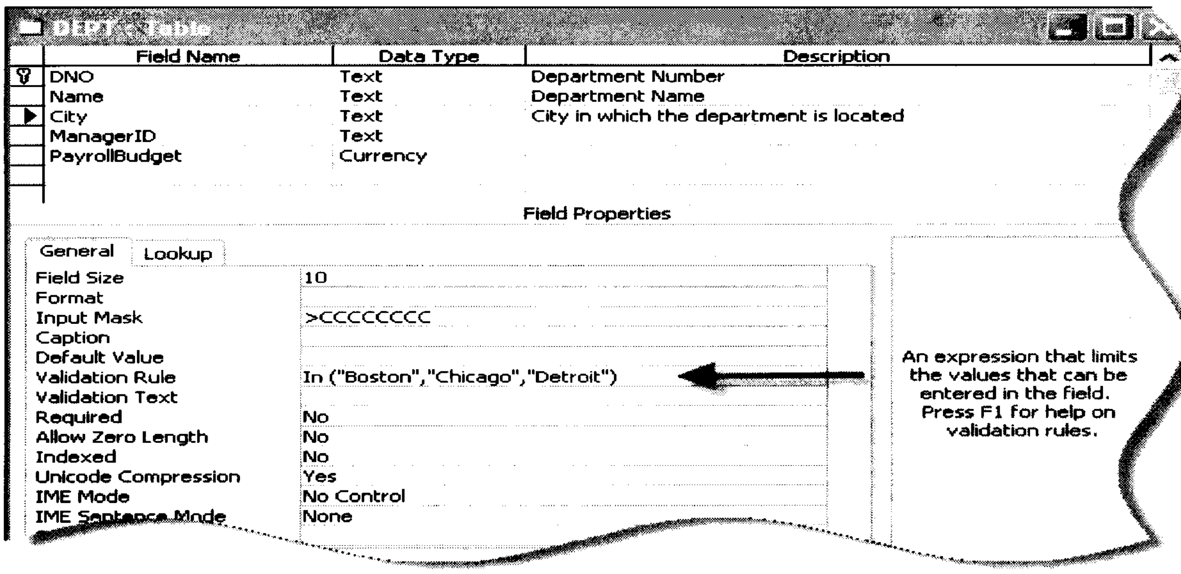
The primary key relation type integrity constraint is easily handled in Microsoft Access by designating the primary key column(s). Closely related relation type integrity constraints arising from candidate keys (such as Social Security Number (SSN) column in our sample EMP table) are handled using indexing as shown in Figure 3. Also, a unique index on *multiple* columns may be specified in Access using an SQL statement such as: CREATE UNIQUE INDEX idx1 ON EMP(SSN, HireDate).

Other kinds of relation type integrity constraints such as Rule 13 must be programmed in Microsoft Access. And, since triggers are not supported, such programming must be attached to every data entry/update form that touches the underlying table. The basic approach is to associate our enforcement code with the **BeforeUpdate** event of the underlying FORM:

```
Private Sub Form_BeforeUpdate(Cancel As Integer)
'
'Rule 13: For department D10 employees hired on the same
'date, the ReviewDate must be identical.

Let Department = Me![DEPTID]
If Department = "D10" Then
  HireDate = Me![HireDate]
  ReviewDate = Me![ReviewDate]

 If IsNull(ReviewDate) Then
  strWhere = "([DeptID] = 'D10') AND "
  strWhere = strWhere & "([HireDate] = #"
  strWhere = strWhere & HireDate & "#) AND "
  strWhere = strWhere & "([ReviewDate] Is Not Null)"
 Else
  strWhere = "([DeptID] = 'D10') AND "
  strWhere = strWhere & "([HireDate] = #"
  strWhere = strWhere & HireDate & "#) AND "
  strWhere = strWhere & "([ReviewDate] <> #"
  strWhere = strWhere & ReviewDate & "#)"
 End If

 'See if updating this row would violate the rule ...
 Let K = DCount("ENO", "EMP", strWhere)
  If K <> 0 Then
   strMsg = "D10 employees hired the same date must "
   strMsg = strMsg & "have identical ReviewDate!"
   MsgBox strMsg
   DoCmd.CancelEvent
  End If

End If
End Sub
```

Rule#	***Static***				***Dynamic***
	Domain Type	Tuple Type	Relation Type	Database Type	
1			✓		
2			✓		
3			✓		
4				✓	
5				✓	
6	✓				
7				✓	
8	✓				
9		✓			
10	✓				
11		✓			
12					✓
13			✓		
14				✓	

Table 1. Classification of Rules 1-14.


DEPT : Table
Field Properties
General
Lookup
An expression that limits the values that can be entered in the field. Press F1 for help on validation rules.

Field Name	Data Type	Description
DNO	Text	Department Number
Name	Text	Department Name
City	Text	City in which the department is located
ManagerID	Text	
PayrollBudget	Currency	

Field Size	10
Format	
Input Mask	>CCCCCCCC
Caption	
Default Value	
Validation Rule	In ("Boston","Chicago","Detroit")
Validation Text	
Required	No
Allow Zero Length	No
Indexed	No
Unicode Compression	Yes
IME Mode	No Control
IME Sentence Mode	None

Figure 1. Using Validation Rules to Assert Domain Type Integrity Constraints.

The referential integrity database type constraints are easily handled in Microsoft Access using the relationship screen where the cascade delete and cascade update triggering actions can also be specified as shown in Figure 4 for Rule 4.

Other kinds of database type integrity constraints must, however, be handled through programming in Microsoft Access. Rule 7, for example, can be enforced by the following VBA code attached to a form that modifies EMP table:

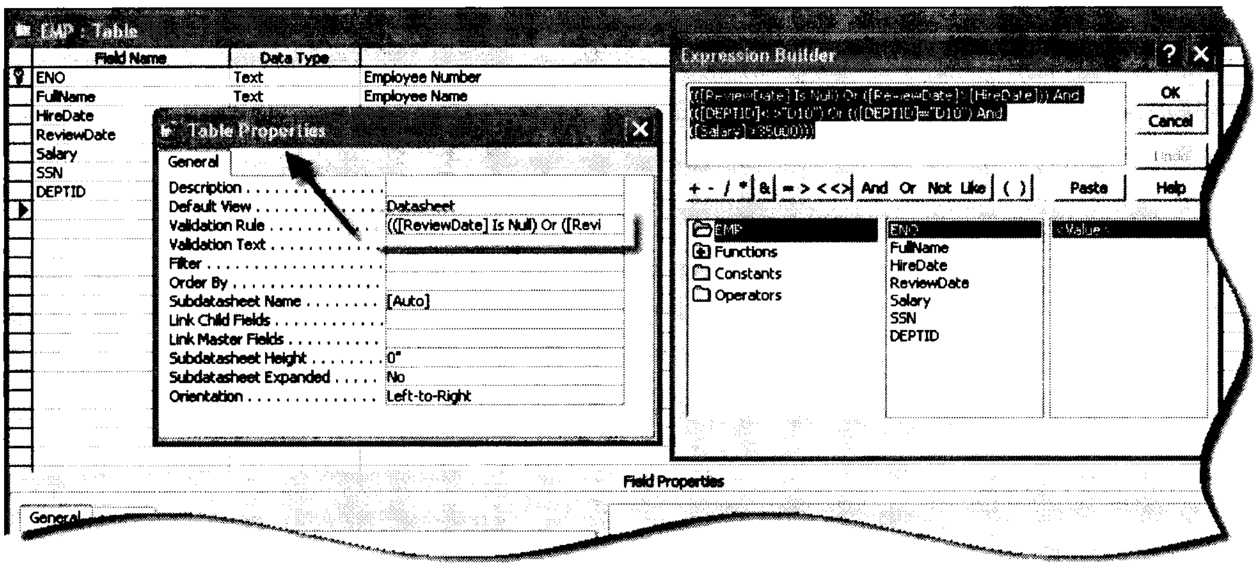
```
Private Sub Form_BeforeUpdate(Cancel As Integer)

'Rule 7: PayrollBudget for each department must be greater
'than or equal to the sum of salaries of employees assigned to
'that department.

Let Department = Me![DEPTID]
Let Salary = Me![Salary]
Let ENO = Me![ENO]

'Get the corresponding department's PayrollBudget ...
PayrollBudget = DLookup("PayrollBudget", "DEPT", "DNO =
'" & Department & "'")
'Get the total payroll of the department excluding
'this employee ...
Payroll = DSum("Salary", "EMP", "DeptID = '" & Department
& "' AND ENO <> '" & ENO & "'")

If PayrollBudget < (Payroll + Salary) Then
 MsgBox "Department's PayrollBudget will be exceeded!"

 DoCmd.CancelEvent
End If
End Sub
```

The referential integrity database constraints are easily handled in Microsoft Access using the relationship screen.

```
Let Department = Me![DEPTID]
Let Salary = Me![Salary]
Let ENO = Me![ENO]

'Get the corresponding department's PayrollBudget ...

PayrollBudget = DLookup("PayrollBudget", "DEPT", "DNO =
'" & Department & "'")

'Get the total payroll of the department excluding'this
employee ...
Payroll = DSum("Salary", "EMP", "DeptID = '" & Department
& "' AND ENO <> '" & ENO & "'")

If PayrollBudget < (Payroll + Salary) Then
 MsgBox "Department's PayrollBudget will be exceeded!"
  DoCmd.CancelEvent

End If
End Sub
```

Finally, Microsoft Access supports enforcing dynamic integrity constraints through programming by making the **OldValue** property available for each data field of the current record. As such, dynamic constraints such as Rule 12 can be handled through code attached to the BeforeUpdate event of the underlying data entry/update FORM:

```
Private Sub Form_BeforeUpdate(Cancel As Integer)
'
'Rule 12: Salaries should not be reduced.

'Get the old salary value or zero (if null) ...
Let OldSalary = Nz(Me![Salary].OldValue, 0)
Let NewSalary = Me![Salary]

If NewSalary < OldSalary Then
MsgBox "Rule 12 has been violated!"
 DoCmd.CancelEvent

End If
End Sub
```



EMP : Table

Field Name	Data Type	
ENO	Text	Employee Number
FullName	Text	Employee Name
HireDate		
ReviewDate		
Salary		
SSN		
DEPTID		

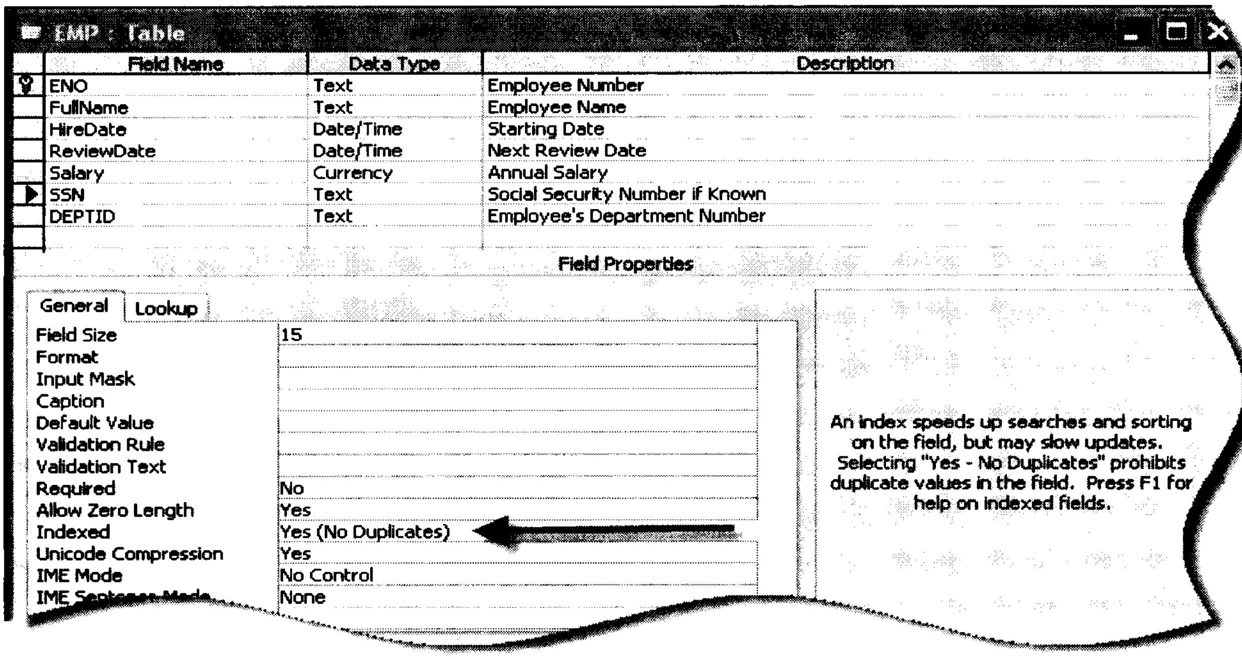
Table Properties — General

Description	
Default View	Datasheet
Validation Rule	(([ReviewDate] Is Null) Or ([Revi
Validation Text	
Filter	
Order By	
Subdatasheet Name	[Auto]
Link Child Fields	
Link Master Fields	
Subdatasheet Height	0"
Subdatasheet Expanded	No
Orientation	Left-to-Right

Expression Builder

```
(([ReviewDate] Is Null) Or ([ReviewDate]>[HireDate])) And
(([DEPTID]<>"D10") Or (([DEPTID]="D10") And
([Salary]>35000)))
```

Figure 2. Using Table Properties to Assert Tuple Type Integrity Constraints.



EMP : Table

Field Name	Data Type	Description
ENO	Text	Employee Number
FullName	Text	Employee Name
HireDate	Date/Time	Starting Date
ReviewDate	Date/Time	Next Review Date
Salary	Currency	Annual Salary
SSN	Text	Social Security Number if Known
DEPTID	Text	Employee's Department Number

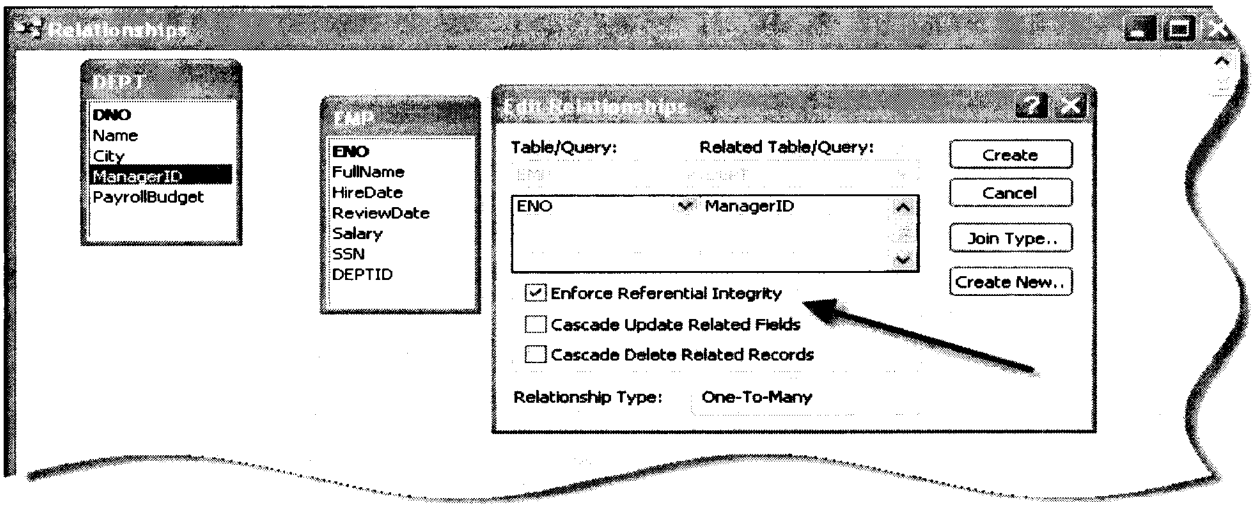
Field Properties — General | Lookup

Field Size	15
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	No
Allow Zero Length	Yes
Indexed	Yes (No Duplicates)
Unicode Compression	Yes
IME Mode	No Control
IME Sentence Mode	None

An index speeds up searches and sorting on the field, but may slow updates. Selecting "Yes - No Duplicates" prohibits duplicate values in the field. Press F1 for help on indexed fields.

Figure 3. Using Indexing to Assert Relation Type Uniqueness Constraints.

As demonstrated in this section, all types of semantic integrity constraints can be specified and enforced in Microsoft Access. For domain type and tuple type integrity constraints, Access supports a declarative approach to specification and provides for automatic enforcement. For relation type and data type constraints other than primary key integrity and referential integrity rules, Access leaves both the specification as well as the enforcement to program logic. Unfortunately, since Microsoft Access does not support the concept of DBMS triggers, program logic developed to enforce integrity. Microsoft Access does not support the concept of DBMS triggers, program logic developed to enforce integrity constraints must be re-created (actually re-used) for each data entry/update form that touches the underlying table. This problem notwithstanding, the lack of support for DBMS triggers in Access means that program logic developed to en force integrity constraints will not be automatically invoked if the user attempts an update through the built-in user interface in Access. As such, to maintain data integrity, the users must be denied access to such useful features as editing a table in datasheet view or running an update query.



Table/Query: EMP   Related Table/Query: DEPT

ENO   ManagerID

[✓] Enforce Referential Integrity
[ ] Cascade Update Related Fields
[ ] Cascade Delete Related Records

Relationship Type: One-To-Many

Figure 4. Using Relationships to Assert Referential Integrity Database Type Constraints

4. CONCLUSIONS

A good database must at all times reflect the real world it is designed to represent. Semantic integrity constraints are logical assertions about the valid states of a database. The importance of the specification and enforcement of semantic integrity constraints has been recognized since the advent of the relational data model. Early papers (Eswaran and Chamberlin, 1975; Hammer and McLeod, 1975) proposed the implementation of constraint checking as an integral subsystem in DBMS software. Unfortunately, however, commercial software adoption of those ideas has been lagging far behind.

Support for declarative semantic integrity constraints (i.e., database assertions) in DBMS software ranging from DB2, Oracle, SQL Server to Microsoft Access remains limited to primary key integrity, referential integrity, and what has been characterized in this paper as domain type and tuple type static constraints. The more complex relation type, database type, and dynamic constraints must be enforced using procedural definition of integrity constraints by triggers. A trigger is a procedure that is automatically invoked by the DBMS in response to specified database events. Although, Microsoft Access supports powerful embedded SQL programming in its VBA procedures, the triggering events are FORM events as opposed to fundamental database events of INSERT, DELETE, or UPDATE on a base table. Therefore, enforcement of complex semantic integrity constraints in Microsoft Access requires "coordination" across data entry/update forms touching a base table, and more importantly demands closing doors to unprotected updates to tables such as by direct datasheet edits or explicit SQL update statements by the user. In this paper, we have presented an approach to teaching categorization of semantic integrity constraints and showing how to implement each of the five categories in Microsoft Access.

5. REFERENCES

Date, C.J. (1990) "A Contribution to the Study of Database Integrity." In *Relational Database Writings 1985-1989*, Edited by C.J. Date. Addison-Wesley, Reading, MA.

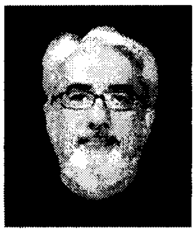
Eswaran, K.P. and Chamberlin, D.D. (1975) "Functional Specifications of a Subsystem for Data Base Integrity." In *Proceedings of the 1st International Conference on Very Large Data Bases (VLDB '75),* Edited by D.S. Kerr. Morgan Kaufmann Publishers, Los Altos, CA.

Grefen, P.W.P.J. and Apers, P.M.G. (1993) "Integrity Control in Relational Database Systems–An Overview," *Data & Knowledge Engineering*, 10(2), pp. 187-223.

Hammer, M.M. and McLeod, D.J. (1975) "Semantic Integrity in a Relational Data Base System." In *Proceedings of the 1st International Conference on Very Large Data Bases (VLDB '75),* Edited by D.S. Kerr. Morgan Kaufmann Publishers, Los Altos, CA.

Türker, C. and Gertz, M. (2001) "Semantic Integrity Support in SQL-99 and Commercial (Object-)Relational Database Management Systems," *VLDB Journal,* 10(4), pp. 241-269.

AUTHOR BIOGRAPHY



Mohammad Dadashzadeh holds a bachelor in electrical engineering, a master in computer science, both from MIT, an MBA, and a Ph.D. in computer and information science from University of Massachusetts. He has been affiliated with University of Detroit (1984-1989) and Wichita State University (1989-2003) where he served as the *W. Frank Barton Endowed Chair in MIS.* In 2003, he joined the faculty of the Decision and Information Sciences Department at Oakland University as Professor of MIS and Director of Applied Technology in Business (ATiB) program. He has authored 4 books and more than 50 articles on information systems and has served as the editor-in-chief of *Journal of Database Management.* His most recent editorial responsibilities include the books entitled *Information Technology Management in Developing Countries* and *IT Education in the New Millennium.*

STATEMENT OF PEER REVIEW INTEGRITY

ISSN 1055-3096