Seamlessness as a desirable aspect of quality for MDE: the contribution of object-relational database stuctures

Patricia Roberts
School of Engineering and Computer Science,
University of Westminster
309 Regent Street,
London W1B 2UW
UK

robertpa@westminster.ac.uk

Abstract—**Where model-driven approaches are used in Information Systems development, as well as transforming models into application code, an important and often overlooked aspect is the transformation into storage schemas for persistent data. Relational database schemas are still being used but these might not be the best quality solutions for persistent data. Object-relational database management systems (ORDBMS) can store persistent data using structures that have more in common with object-oriented application code structures. Seamless transformations may have the quality that is desirable for model-driven approaches. In this paper we demonstrate the advantages of seamless transformations. We show object-relational structures that contribute to seamlessness and the implications for model-driven approaches such as Model-Driven Engineering.**

Keywords-component; object-relational databases, model-driven engineering, transformations, quality

I. INTRODUCTION

In model-driven approaches emphasis is placed on transforming models into application code. However, another important transformation is that into a storage schema for the persistent data in a system. Relational database technology is still being used for storing the persistent data of an application, even when the original conceptual model is object-oriented. But if the target of a transformation is a relational DBMS, a radical transformation is required from an object-oriented model. Although this type of transformation is common, it cannot be achieved without some loss of semantics because of the paradigm difference between OO and relational, resulting in impedance mismatch problems, such as described by Ambler [1]. However, object-relational databases (ORDBMS) can store persistent data using structures that have more in common with an object-oriented conceptual model and OO application code structures. If ORDBMS technology is used, transformations from the conceptual model can be more 'seamless'. If seamless transformations can provide the quality that is desirable for model-driven approaches, then transformation into ORDBMS structures should be considered.

In this paper we present an argument for the consideration of seamlessness as a desirable quality in transformations. We show how some object-relational database features can contribute to seamlessness and we discuss some of the implications that this has for transformations of the persistent data in an application, such as could be used in model-driven development. To illustration the concept of seamlessness in transformations, we present one feature of an object-oriented conceptual model, a generalization hierarchy, and examine options for transformations that are available. We then show that some of these transformations exhibit the quality of seamlessness, while others do not.

II. BACKGROUND

The introduction of object-relational databases (ORDB) brought new structures to add to traditional relational tables. Stonebraker [2] described object-relational as "The Next Great Wave" anticipating that the new technology would revolutionize database design. Other writers, such as Brown [3] present ways to use the new ORDB features and books on databases design, such as Connolly and Begg [4], present ORDB as an option for implementation of database designs. However, while ORDB has been available for many years, the impact of these structures on the quality of database designs has not yet been established. A number of researchers and practitioners have evaluated ways of transforming associations [5], aggregations [6, 7] and hierarchies [8, 9] into object-relational structures. In addition, a number of articles have tried to assess the quality of object-relational design [10, 11]. However, the focus of this research is on the simplicity of the designs and has not considered whether the quality of seamlessness is present in the transformations.

The term Model-Driven Engineering (MDE) is used to describe the development of software through the automatic transformation from conceptual models through to concrete implementations. At the heart of MDE is the use of models to

describe complex systems at different levels of abstraction and from different perspectives and the use of automatic transformations of the models. If a model, such as a class model, is to be used for an automatic transformation, then searching questions will be asked about the quality of the models, a point made by France and Rumpe [12]:

"If models are the primary artifacts of development then one has to be concerned with how their quality is evaluated. Good modeling methods should provide modelers with criteria and guidelines for developing quality models. ... The reality is that modelers ultimately rely on feedback from experts to determine "goodness" of their models."

Mens et al [13] identified the characteristics of different kinds of transformations in model-driven development. Using their taxonomy we would say that this paper is addressing:

- Vertical transformations from an abstract to a more concrete level
- One source models and (potentially) multiple target models
- The language of the source model is the UML class diagram
- The language of the target models is SQL

Much of the research focus on vertical transformations has concerned the transformation from abstract model to program code (the transformation labeled 1 in Fig.1). Further extensive work has established transformations from program code to database structures to preserve persistent object (the transformation labeled 2 in Fig.1). Both these transformations are important in establishing robust model-driven development. Here we are focusing on another kind of transformation (the transformation labeled 3 in Fig.1), where the database structures in question are object-relational structures. However, the notion of seamlessness could just as easily be applied to other transformations like 1 and 2.

In summary, much research has concentrated on the quality of models and on transformations from models into application code. Although object-relational database features have been evaluated, the transformations into object-relational database

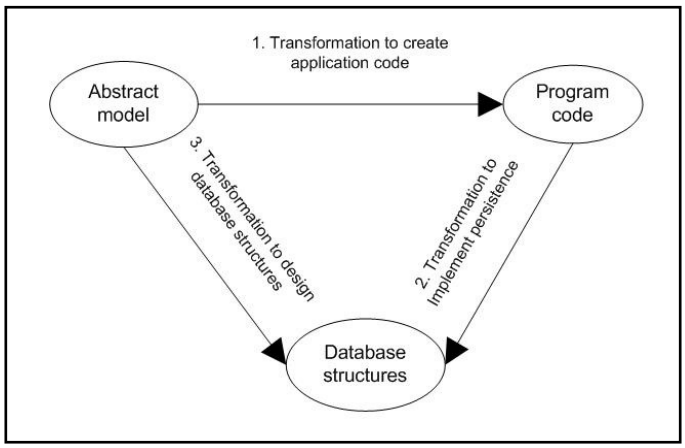

Fig. 1 Transformation directions

should focus on seamlessness if they are to be used as part of any model-driven process.

III. SEAMLESSNESS AS A QUALITY FACTOR

One reason for a designer to use an ORDBMS would be to bring the database design closer to the OO analysis product that it is derived from, for example a UML class diagram [14]. If a suitable design could be produced using simple relational tables, then that might be considered preferable, certainly in terms of maintainability, as the relational model is usually simpler. The UML class diagram is often used to model the conceptual objects in a system and can be taken forward to become a database design. However, UML class diagrams may contain aggregation, hierarchies, directional navigation using pointers, multi-valued attributes etc., which cannot be directly implemented using relational tables, but could be implemented using object-relational features. One of the key motivations for the introduction of object-relational structures into SQL was to address the impedance mismatch between OO applications and relational databases [2]. Seamlessness is at the heart of the motivation for object-relational databases, with the reduction of the impedance mismatch as a key aim, which is why it should be considered as an aspect of quality. If the transformation from UML to ORDBMS schema is completely seamless then the two representations will be identical. The more differences there are between the two, the less seamless the realization becomes. To assess the seamlessness of transformations we need to know how the object-relational feature is created, but we also need to know how it would be used, for example when creating, reading, updating and deleting information (commonly called CRUD operations). By examining the way features can be created and used we can compare different transformations to assess their seamlessness.

IV. OBJECT-RELATIONAL STRUCTURES

In this section we examine some object-relational features that are part of the SQL:2008 standard [15] (the current standard at time of writing that supersedes all previous versions). Most of these features; user-defined types, collection types, row type, type and table hierarchies and REF types were part of the SQL:1999 [16] revision of the standard, with the MULTISET added in SQL:2003 [17] together with some other minor changes. Since SQL:2003 there have been no significant changes to the object-relational features in the standard.

A. User-defined types (UDT): distinct types

Within SQL:92 and earlier versions of the standard, certain built-in data types were defined and could be used to specify the set of values for a column of a table. Distinct Types are an extension of this idea, to specify a set of values as having a distinct meaning.

B. User-defined types (UDT): structured types

A Structured Type is a particular kind of UDT that has an internal structure. Once a Structured Type has been declared it can be used within a column definition, just as other UDTs. The constituent parts of structured types can be referenced

separately in relational operations. The manipulation of User-defined Types is similar to the way the way that objects are manipulated within an O-O programming language. The data within a User-defined Type are encapsulated in a similar way to an object, in that the contents cannot be directly manipulated, but are hidden from the outside.

C. Collection types

Collection Types are structures in which there are a number of elements of the same type. There are two collection types defined in SQL: Arrays were included in the SQL:1999 standard [16] and Multisets were introduced in SQL:2003 [17]. The difference between Arrays and Multisets is that Arrays have a notion of ordering within the collection whereas Multisets do not. Some new operations on Multiset were included in SQL:2003: UNNEST, COLLECT, FUSION, INTERSECTION, CARDINALITY and SET. Kulkarni [18] has shown that these operations are useful for manipulating the data within a Multiset.

D. Row types

The concept of rows was always implicit in the definition of a table in SQL Data Definition Language (DDL) but, with the introduction of Row definition in SQL:2003, came the possibility of defining a Row Type separately from a table definition. The Row itself is not an OO construct, but it can be used in a collection without the underlying concept of representing objects.

E. Type hierarchies

The OO concepts of generalization and specialization hierarchies are reflected in the ability in SQL:2008 to define types as hierarchical structures. Sub-types can be defined that inherit attributes and methods from their super-type.

F. Typed tables

A Typed Table is a new kind of table that is based on a UDT. A significant difference from a traditional table is the REF clause that is required to be defined on all Typed Tables, except where they inherit the clause (see Appendix A, option 1 for an example). The REF clause creates another column for the table that is termed a self-referencing column. This gives the row a unique identity that can be referenced by other components in the environment. This is a way in which the rows of Typed Tables have characteristics of objects in an OO system. However, this identity is different from the object identity that the object will have when instantiated in an OO programming environment. From an OO viewpoint a Typed Table can be seen as a mechanism for storing objects. Each row of the table would store one unique object of the UDT defined for the table. The fact that the objects are stored in rows is not relevant to the OO developer.

G. Table hierarchies

A Type Hierarchy creates a structure but does not create any storage mechanism within the database. If the objects from the Type Hierarchy are to be stored in the database, they can be used as part of a Table Hierarchy or as the type of a table column. When Melton [8] describes the OR extensions that were introduced with the new SQL:1999 standard, he compares the different approaches to database design that can be employed using these features. Melton focuses on the different ways that hierarchies can be implemented in object-relational databases, using traditional relational tables, defining type hierarchies and using them in table columns, or creating full typed table hierarchies. The benefit of the third approach, using Typed Tables, is that it can take advantage of OO design, providing a seamless crossover to programming in languages such as Java and would be more familiar to OO designers. It would produce what is essentially an OO design, but with the storage and querying capabilities of a relational database. In order to achieve this, the familiar SQL queries of SQL92 have been extended to allow querying over the hierarchies.

H. REF types

REF is a built-in data type that was introduced in SQL:1999 [16], which is crucial to the object-relational features. REF is a data type in an object-relational database that is similar to an object reference in an OO programming language. Each REF is a unique reference to some ‘object’ in the database, which can be thought of as a pointer to the ‘object’, although the ‘object’ is held as a row in a table. In many ways REFs are similar to the foreign keys of traditional relational database. However, differences emerge with the way that REFs can be used to navigate through the data. Using the term ‘*DEREF*’ a query that accesses one table can use a REF to find data in another table. This way of ‘navigating’ through a database is a departure from the relational model.

V. IMPLICATIONS FOR MDE

In any model-driven development, where the model is to be used for an automatic transformation, we need to be concerned about the quality of the models and the quality of the transformations. Seamlessness is a candidate for a quality measure of transformations. Here we present one feature of an object-oriented conceptual model, a generalization hierarchy, and examine the different options for transformations. The generalization hierarchy is a concept of abstraction which is central to OO design and it is used extensively in OO programming languages. A generalized class can be created to capture the commonality between classes of objects. In a class hierarchy the general class (or super-class) contains the common attributes and operations and the more specialist classes (or sub-class) inherit attributes and operations from their super class.

However, designers and programmers may have different motivations for using a hierarchy in a design or program. The inheritance of attributes and operations through a hierarchy is seen as a major way to improve reuse of programming code within a system. By defining a super-class, common features of the different classes can be defined once and reused within the sub-classes. The case for using a class hierarchy as part of a design, because it meets the conceptual needs of the model, is distinct from the use of a hierarchy to allow for reuse of code. The reasons for using a class hierarchy may be pragmatic or conceptual but they are now an accepted part of

OO design. While class hierarchies are part of the language of conceptual models (for example, as part of the UML class model) and have become an accepted part of OO programming languages, the use of hierarchies in databases is not universal. As the relational model has become the dominant model, the need to find ways to represent hierarchies in databases has become important.

When addressing the problem of transforming inheritance hierarchies from a conceptual model into a database representation there are two common approaches. Hierarchies can be flattened into relational tables or they can be implemented as Type and Table hierarchies in an object-relational database. Although these are not the only approaches that can be used they are the most common.

The flattening of hierarchies into a single relational table (or into several tables linked by foreign key constraints) is a technique that has long been used because of the need to store data from hierarchies when a relational database is the technology available. A single relational table can be created that contains the accumulated attributes of the superclass and all subclasses. Additionally, further attribute or attributes are added to distinguish between the different classes in the hierarchy. A Rational Software whitepaper [19] describes a set of rules for mapping from classes to relational tables. In mapping an inheritance hierarchy they state that *"The corresponding data model specifies 2 tables and an identifying relationshi*p". Later it presents an example of an inheritance hierarchy that is mapped to a single table with nullable attributes that represent the data of the subclass. The statement at the end indicates that, although no decision making is indicated in the process, some choices about the form of the mappings have to be made: *"In most cases, the data analyst makes decisions about merging tables based on optimising the database for data access."*.

Lodhi and Ghazali [20] describe mapping inheritance hierarchies using foreign keys, a technique also advocated by Ambler [21]. Lodhi and Ghazali describe a vertical mapping strategy:

"In vertical mapping, each class of the inheritance hierarchy, whether abstract or concrete, is mapped to a separate table. To maintain the inheritance relationship between parent and child classes, primary key (OID) of the parent class is inserted in the child classes as a foreign key."

It is worth noting that here the writers wrongly equate a primary key with an object-identifier (OID). Although these concepts are both ways to identify objects, and they are often confused, in this context the key is a traditional primary key/foreign key pair used to link together the tables.

Eder and Kanzian [22] examine the "decision space for designers" regarding ways to implement inheritance hierarchies and compare seven different ways that they can be transformed into relational and object-relational structures. Their analysis of the performance of the seven alternatives for implementing hierarchies shows marked differences between them, when implemented in an object-relational DBMS (Oracle 9i).

To illustrate the notion of seamlessness in transformations, we shall take two from the many options for transforming hierarchies and compare their properties. For this experiment we need a conceptual model containing a hierarchy, to be our source model. Fig.2 shows a very simple class model, containing a hierarchy, that we can use. This contains a super-class called "Member" (of a library) with two sub-classes, "Employee" and "Student". Operations have been omitted from the diagram.

The simple hierarchy in Fig.2 can be transformed in many different ways, as discussed above. We shall choose two contrasting approaches: option 1 is a transformation into an object-relational design using a hierarchy of types and typed tables; option 2 is a relational transformation with one table containing all the attributes of the classes in the hierarchy. The SQL:2008 code to create these two structures is shown in Appendix A.

Now we need some principles for comparison of the seamlessness of the two transformations. As seamlessness is not yet well established as a quality aspect of transformations, we must begin with some basic assertions. We assert that a seamless transformation would have certain characteristics:

1. Similarity: the representations of the source and target structures are similar. For transformations from a class model to a database structure, a seamless transformation would contain similar patterns and encapsulate the same concepts.
2. Correspondence: there is a one-to-one mapping between the source and target models. For example, one class on a class model would result in one structure in the database model.
3. Reversibility: it would be possible to use the target model, such as a database structure, to derive or 'reverse engineer' the source model.

These three characteristics of seamless transformations; similarity, correspondence (one-to-one) and reversibility are not formally defined here. The informal definitions used here are such that they can be used in comparisons of transformations by a domain expert. More formal definitions

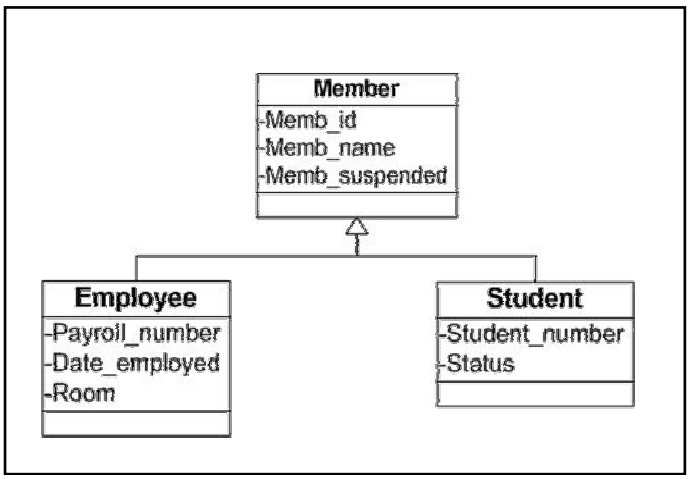

Fig.2 Realizations for a generalization hierarchy

of the three characteristics could be developed and would be required for comparisons to be automated. This point is taken up at the end of this paper.

If we examine the SQL code in Appendix A we see that the first option creates structures directly analogous to the structure of the hierarchy in the class model. The creation of sub-types uses the word '*UNDER'* to designate it as a being part of a hierarchy as in the phrase to create Employee: "*CREATE TYPE Employee UNDER Member*" The second option creates a single relational table. While the structure is used to represent the hierarchy, the code to create it has none of the characteristics of a hierarchy.

When we examine the second question, we find that option 1 creates a table for each class in the hierarchy and no additional tables, so it is indeed a one-to-one mapping. Option 2creates one relational table, regardless of the number of classes in the hierarchy and therefore is not a one-to-one mapping.

Finally, we have the question of whether we can reverse-engineer the model from the database structure. Examination of the create statements for option 1 would clearly indicate that a class hierarchy was the source structure. This would not be evident on examination of the create table statements for option 2.

This comparison of the two options for transformation of hierarchies clearly indicates that one exhibits the quality of seamlessness and the other does not. However, we have only presented here a small example of the assessment of seamlessness in transformations. Further work has been done to examine many object-relational transformations [23], not only for hierarchies, but for other structures such as associations and aggregations. This work shows that the principles for comparison of transformations used here can be applied more widely and are valuable in distinguishing between the qualities of transformations.

VI. Conclusions and Future Work

When object-relational databases are used, the multitude of options for transformations that are available could present problems for model-driven approaches. It may be difficult to evaluate the quality of transformations using aspects such as simplicity. However, if seamlessness is a quality that is sought in transformations, we may find that certain options become more attractive. It is possible to analyze the seamlessness by using the characteristics presented here. By examining the seamlessness of transformations we can narrow down the options and enable MDE and other model-driven approaches to be used when considering options for transforming the persistent objects in a system into database representations.

The informal definitions of seamlessness used in these comparisons are sufficient for expert evaluations of these transformation options. To develop the work further, the characteristics of seamlessness: similarity, correspondence and reversibility, could be formally defined. This would facilitate more automation of comparisons and could lead to the development of seamlessness metrics. Further work is also needed to assess the value of seamlessness in reducing the impedance mismatch between object-oriented and database systems.

VII. ACKNOWLEDGEMENTS

This research is derived from a PhD thesis on structural transformations in object-relational design [23]. The author thanks the following persons for their valuable contributions to the work: David Bowers, Mike Newton and Kevin Waugh of The Open University. We also extend thanks to the anonymous reviews of this paper for their time and comments and to Angelos Stephanidis for feedback on drafts of the paper.

VIII. References

[1] Ambler S 2009 The Object-Relational Impedance Mismatch. Available from: http://www.agiledata.org/essays/impedanceMismatch.html Accessed: Jan 2008

[2] Stonebraker MR, Brown P. 1999 Object-relational DBMSs: Tracking the next great wave. San Francisco, CA: Morgan Kaufmann.

[3] Brown P. 2002 Developing Object-Relational Database Applications. Available from: http://www.ibm.com/developerworks/data/library/techarticle/0206brown/0206brown1.html

[4] Connolly T, Begg C. 2005 Database Systems: A Practical Approach to Design, Implementation and Management (4th Edition). Harlow, Essex: Addison Wesley.

[5] Soutou C. 2001 Modeling relationships in object-relational databases. Data Knowl Eng. 36(1): p. 79

[6] Marcos E, Vela B, Cavero JM, Caceres P. 2001 Aggregation and Composition in Object-Relational Database Design. ADBIS (5th east european Advances in Databases and Information Systems). Vilnius, Lituania: Springer.

[7] Rahayu JW, Taniar D. 2002 Preserving aggregation in an object-relational DBMS. In: Yakhno T, editor. 2nd International Conference on Advances in Information Systems. Izmir, Turkey: Springer-Verlag Berlin.

[8] Melton J. 2003 Advanced SQL:1999 Understanding Object-Relational and Other Advanced Features. San Francisco, CA: Morgan Kaufman.

[9] Roy J 2003 Using the Node Data Type to Solve Problems with Hierarchies in DB2 Universal Database. Available from: http://www-106.ibm.com/developerworks/db2/library/techarticle/0302roy/0302roy.html

[10] Baroni A, Calero C, Piattini M, Abreu F. 2005 A Formal Definition for Object-relational Database Metrics. 7th International Conference on Enterprise Information Systems. Porto, Portugal.

[11] Calero C, Sahraoui HA, Piattini M, Lounis H. 2001 Estimating object-relational database understandability using structural metrics. In: Mayr HCLJQGVP, editor. 12th International Conference on Database and Expert Systems

Applications (DEXA). Munich, Germany: Springer-Verlag Berlin, p. 909.
[12] France R, Rumpe B. 2007 Model-driven Development of Complex Software: A Research Roadmap. 2007 Future of Software Engineering. Minneapolis, Minnesota, USA: IEEE Computer Society, p. 37.
[13] Mens T, Czarnecki, K. and Van Gorp, P. 2004 A Taxonomy of Model Transformations. 04101. Available from: http://drops.dagstuhl.de/opus/volltexte/2005/11/pdf/04101.SWM2.Paper.pdf
[14] Booch G, Jacobson I, Rumbaugh J. 2000 Unified Modeling Language Specification version 1.3. Available from: www.OMG.org
[15] ISO/IEC. 2008 Standard 9075:2008 SQL standard definition. Available from: www.iso.org.
[16] ISO/IEC. 1999 Standard 9075:1999 SQL standard definition. Available from: http://www.iso.org.
[17] ISO/IEC ISO. 2003 Standard 9075:2003 SQL standard definition. Available from: http://www.iso.org.
[18] Kulkarni K 2003 Overview of SQL:2003. Available from: www.wiscorp.com/SQL2003Features.pdf Accessed: Sep 2007
[19] RationalSoftware 2000 Mapping Object to Data Models with the UML. Available from: http://www.uml.org.cn/oobject/tp185.pdf Accessed: Oct 2005
[20] Lodhi F, Ghazali MA. 2007 Design of a simple and effective object-to-relational mapping technique. Proceedings of the 2007 ACM symposium on Applied computing. Seoul, Korea: ACM.
[21] Ambler SW. 2003 Mapping Objects to Relational Databases. Available from: http://www.agiledata.org/essays/mappingObjects.html. Accessed: Jan 2005
[22] Eder J, Kanzian S. 2004 Logical design of generalizations in object-relational databases. 8th East European Conference of Advances in Databases and Information Systems. Budapest, HUNGARY: Magyar Tudomanyos Akademia, p. 16.
[23] Roberts P 2008 Criteria for assessing object-relational quality. Available from: http://computing-reports.open.ac.uk/index.php/2008/200817
[24] Roberts P. 2010 Structural transformations in object-relational design: a framework for improving quality. PhD thesis, Milton Keynes: The Open University.

IX. APPENDICES

A. *SQL code to create generalizations*

Option 1: A hierarchy of typed tables

```
CREATE TYPE Member AS (
Memb_ID INTEGER,
Memb_name Personal_name,
Memb_suspended BOOLEAN DEFAULT FALSE)
NOT INSTANTIABLE NOT FINAL
REF IS SYSTEM GENERATED;

CREATE TYPE Employee UNDER Member AS (
Emp_payroll_number INTEGER,
Emp_date_employed DATE,
Emp_room CHAR(4) )
INSTANTIABLE NOT FINAL;

CREATE TYPE Student UNDER Member AS (
Student_number INTEGER,
Student_status VARCHAR(20))
INSTANTIABLE NOT FINAL;

CREATE TABLE tbl_Member OF Member(
REF IS Memb_ref SYSTEM GENERATED,
Memb_ID WITH OPTIONS CONSTRAINT pk_Member
PRIMARY KEY (Memb_ID));

CREATE TABLE tbl_Employee OF Employee
UNDER tbl_Member( );
CREATE TABLE tbl_Student OF Student
UNDER tbl_Member( );
```

Option 2: One relational table containing all attributes

```
CREATE TABLE tbl_Member(
Memb_ID INTEGER NOT NULL,
Memb_name Personal_name,
Memb_suspended BOOLEAN DEFAULT FALSE,
Memb_is_employee BOOLEAN,
Memb_is_student BOOLEAN,
Memb_emp_payroll_number INTEGER,
Memb_emp_date_employed DATE,
Memb_emp_room CHAR(4),
Memb_student_number INTEGER,
Memb_student_status VARCHAR(20),
CONSTRAINT pk_Member PRIMARY KEY (Memb_ID));
```