Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/080233

Towards a UML Profile to Relational Database Modeling

*Chih-Min Lo** *and Hsiu-Yen Hung*

Department of Digital Media Design, Hwa Hsia Institute of Technology, New Taipei 235, Taiwan

Received: 5 Apr. 2013, Revised: 6 Aug. 2013, Accepted: 8 Aug. 2013
Published online: 1 Mar. 2014

Abstract: Database management systems provide a mechanism that enables software application systems to manipulate data from a database. Although object-oriented technology is widely used in the development of software systems, relational database management systems remain the dominant database technology. Database modeling enables software developers to design the database during the system analysis and design phase; however, most approaches focus only on designing database schemas, without considering models for the retrieval of data. A comprehensive database model would provide a solid base on which software developers could organize data for storage and retrieval. Unified modeling language (UML) is a general purpose modeling notation and this paper proposes a UML profile for modeling database retrieval to overcome the inadequacies found in current methods. The proposed model provides views of the database outlining query operations to enable the automatic generation of more comprehensive code in the model-driven development of enterprise information systems. We include examples to demonstrate the feasibility of the proposed method and its advantages over existing database modeling methods.

Keywords: Software engineering, database modeling, model-driven software development, UML

1 Introduction

Information systems are widely used as an efficient means to process huge volumes of data. Currently, most enterprise information systems (EISs) employ object-oriented programming languages for the implementation of the application layer and the storage of data within a database management system. Although object-oriented database management systems have succeeded in obtaining a share of the market, relational database management systems (RDBMS) remain the dominant technology [6, 18]. The main purpose of an EIS is to provide an interface enabling users to retrieve data from a database; therefore, many RDBMS vendors add structured query language (SQL) [11] as a standard in their database management systems [17]. SQL is a declarative database manipulation language that enables users and applications to access data from a database. Application systems normally retrieve data through an SQL SELECT operation; therefore, modeling the database so that SQL code can be generated is crucial to the development of the EIS. A comprehensive database model should include models for data operations capable of illustrating database schemas and indicating database retrieving operations.

Unified modeling language (UML) is a general purpose language for visual modeling. Although it is probably the most widely used modeling language in software development [24], it cannot satisfy all information modeling needs. For this reason, UML 2.0 was developed to provide two kinds of extension: a heavyweight extension method based on the direct modification of the UML metamodel; and a lightweight extension that allows system analyzers to adapt the UML semantics without having to change the UML metamodel [14, 21]. UML profile is a lightweight extension mechanism for customizing UML models within a particular domain [20]. UML profiles are defined in terms of three basic mechanisms: stereotypes, tagged values, and constraints [9]. These three basic mechanisms are used to denote and limit new elements in the models; however, the UML standard and newly proposed profiles in later revisions do not adequately address the modeling of database operations [18]. In 2005, the Object Management Group (OMG) issued a request for proposal (RFP) for a UML profile in the area of database modeling [15]. In recent years, although the OMG has released several UML profiles for application in specific areas, database operation modeling has not been

* Corresponding author e-mail: cmlo@go.hwh.edu.tw

addressed yet [18]. Therefore, it is necessary to develop a UML profile for database operation modeling.

The OMG proposed model-driven architecture (MDA) as a software development framework emphasizing model-based abstraction and automated code generation. MDA separates a single model into three models: a computation independent model (CIM), platform independent model (PIM), and platform specific model (PSM) [12]. A CIM is similar to a business model that focuses on capturing domain concepts and acquisition requirements. A PIM is a logic model that concentrates on the configuration of the architecture. A PSM greatly resembles a physical model focusing on interoperability and the implementation of coding [22]. Model-driven development (MDD) is based on the MDA framework, representing a new paradigm in the field of software systems development. The successful development of model-driven software is based on complete models capable of addressing all information regarding the properties in the transformation of models. In fact, a transformation from PSM models into executable code is the ultimate purpose in MDD. To make use of the MDD approach, the information represented by models must be consistent, integrated, and computable, enabling automatic transformation from the model into an executable system [19]. Therefore, determining how to create a complete model containing all of the necessary information for transforming models into SQL code is a critical factor in the development of model-driven EISs.

A comprehensive definition of the entire system is a prerequisite for MDD; however, results from surveys on the use of UML has shown that most software practitioners focus only on structural modeling and ignore the operational modeling [1,5], which hampers the practical application of the MDD. Most existing database modeling methods consider the modeling of data schema rather than the modeling of data retrieval. A number of proposed methods take into account the importance of information or data retrieval [2,3,10,18]; however, these methods are only capable of pointing out the framework of data retrieval operations and are unable to produce a comprehensive database model. A complete database model should be able to support the transformation of models into SQL SELECT operation code and illustrate both the structure of data and the relationship of the calculation.

To overcome these problems, this paper proposes a UML profile for designing database retrieval models. The proposed method, named "database retrieval modeling profile", is based on the mechanism of UML profile extensions. The proposed database retrieval modeling profile defines a set of stereotyped classes, one stereotyped attribute, and a set of stereotyped relationships. The stereotyped classes are used to indicate the database table schema, and provide views of the schema and query results. Stereotyped attributes are used to denote columns owned by a table, a view, or a dataset from a database query operation. Stereotyped relationships are used to illustrate the calculation of set operations between two tuples in relational algebra and object constraint language (OCL) is also used to define rules for verifying the quality of elements in database models [13].

The remainder of this paper is organized as follows. In Section 2, we briefly describe various database modeling methods and discuss related work in detail. Section 3 provides the specifications of a UML database retrieval modeling profile and the rules used to verify elements of the model. We present a case study in Section 4. Finally, in Section 5 we present our discussion and conclusions, listing a comparison matrix table to demonstrate the feasibility of our method and its superiority over existing methods.

2 Related Work

Database models are used to exhibit the structure and relationships of data and provide a tool for communication among the members of a development team. In the past few decades, many researchers have proposed database modeling methods such as entity-relationship (ER) modeling [19], information engineering (IE) data modeling notation [6], and integration definition for information modeling (IDEF1X) [7]. These methods are only considered database schemas for the purpose of storing data. Today, UML is becoming increasingly popular as a modeling language, widely used in database modeling.

In recent years, various researchers have proposed methods of database modeling to overcome the inadequacies in existing methods. Song et al. (2007) proposed a database modeling method focusing on dynamic operation modeling. They used frames of UML sequence diagrams to construct database operations such as INSERT, UPDATE, DELETE, and SELECT operations [18]. This paper focuses only on database retrieval operations and does not address the issue of data maintenance. Select operations are also called query operations, representing the most important operations in a database management system, tasked with retrieving data from a database. Song et al. used a UML sequence diagram to model query operations using low level processes. Although their model is capable of representing the processes of query operations, their model is unable to draw the structure of the results returned from query operations. Understanding the structure of data is necessary for the development of information systems.

Databases contain two schemas Table and View. Table is a physical schema for storing data; View is a virtual table, which does not have an actual schema or data, constructed by a database query operation. In database modeling, this should include the structure of data, derived tables and the relationships associated with relational calculus. Today, most software developers use

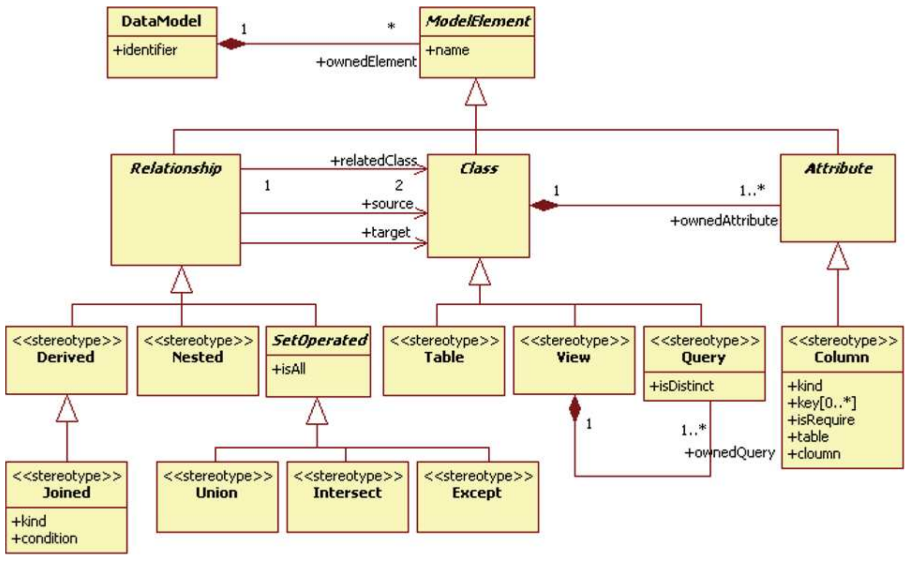


Fig. 1: Overview of database retrieval modeling profile.

UML class diagrams to model database models. Nevertheless, most of these focus only on how to design a database table schema. Ambler (2003) [2] and Gornik (2003) proposed UML data modeling profiles for relational database modeling [10]. These two modeling methods are similar in their use of the UML extension mechanisms to define a UML profile for relational database modeling. Their respective UML profiles define a set of stereotyped classes such as Table and View to state database schemas. They use an attribute of class to denote a simple column for the database Table or View. Computed columns are represented by OCL expressions and dependency relationships are used to illustrate the relationship of derivation. Unfortunately, these two methods are only capable of drawing dependency or denoting the relationships of relational calculus. Database query operations may include many forms of relational calculus such as join, union, intersection, set difference, and nested subquery.

OCL is a modeling language used to specify the constraints of elements within a model. Balsters (2003) proposed a database modeling method using UML class derivation and the OCL framework to model relational database views using the OCL-based approach [3]. Balsters used simple UML class diagram notation and complex OCL expressions to denote a database view. This method uses Class to show a database Table or View, in which only a schema is displayed. It also uses Attributes to indicate simple columns and OCL expressions to describe computed columns. Armonas and Nemuraitė proposed a method using OCL and based on patterns for transforming models into SQL SELECT code [23]. Although these methods are capable of representing a database view using UML and OCL, it fails to state the database view with UML class diagrams. In addition, it is not a visual modeling language and does not lend itself well to communication among members of development teams or end users.

The previously described methods are incapable of providing a comprehensive database model. The derived models do not provide the means with which the members of software development team can communicate, or obtain sufficient information for the development of model-driven information systems. Therefore, this study proposes a UML profile to define a set of stereotypes for the specification of new notation associated with UML class diagrams. The proposed method is capable of representing database query results and the structure of relational-calculus using graphic notation. We proposed the UML profile only to discuss the means of modeling database retrieval, rather than deal with the issue of data modeling, because other researchers have proposed good methods to deal with this.

3 Modeling Approach

This paper proposes a modeling method to define UML profile packages for database retrieval modeling. The proposed method enables the developers of information systems to design database information retrieval models, and automatically convert these models into SQL code using a code generator. UML profiles are defined using stereotypes, tagged values, and constraints, applied to specific model elements, such as classes, attributes, operations, and relationships. A stereotype is one of three types of extensibility mechanisms in the UML, allowing the extension of the UML vocabulary to derive new model elements from existing ones. A tagged-value combines a tag and a value to provide supplementary

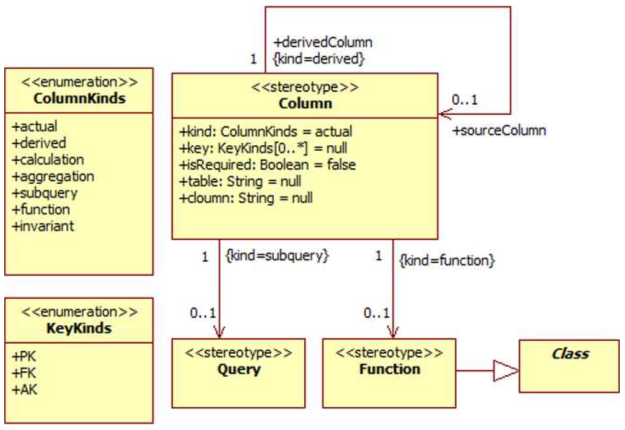


Fig. 2: Metamodel of Table, View and Query.

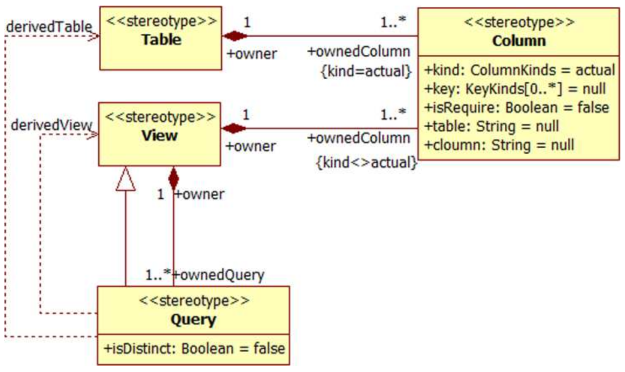


Fig. 3: Metamodel of Column.

Table 1: List of tagged values for Column tag: *kind*

Value	Description
actual	This value represents a physical column, probably owned by a *Table* class.
derived	This value denotes a none physical column, probably owned by a *View* class, and it is bound to another column which owned by a table or a view.
calculation	This value shows a column formed by a calculation expression.
aggregation	This value indicates a column comprising aggregates, such as sum, average, count, minimum, and maximum.
subquery	This value states a column formed by a sub query expression.
function	This value represents a column formed by a function result.
invariant	This value shows a column formed by the column default value.

information that is attached to a model element. A tagged value can be used to add properties to any element in the model. Constraints enable users to refine the semantics of elements in a UML model and refine model elements by expressing a condition or a restriction in a textual statement to which the model element must conform.

Figure 1 shows an overview of the metamodel of the database retrieval modeling profile. This paper defines three kinds of stereotypes inherited from metaclass: class, attribute, and relationship. The stereotypes Table, View, and Query are three classes used to state a database table schema, database view schema, and set of results related to a database query operation. A stereotype Column shows attributes owned by a Table, View, or Query. The relationship includes six different stereotypes: Derived and Joined represent the relational algebra, and stereotypes Union, Intersect, Except and Nested indicate tuple relational calculus.

3.1 Specification of Class stereotypes

Figure 2 shows a specification of the stereotypes Table, View, and Query metamodel. These three stereotypes belong to the element Class in a UML model. In database modeling, stereotype Table represents a table schema stored in a relational database. A table is a main schema containing one or more actual columns with unique names. Table is a mechanism for storing data, and holding physical data records. Another stereotype View is used to denote a database view schema in a database model. The difference is that a View owns one or many actual columns. In a model, View contains at least a Query stereotype class. Stereotype Query denotes the structure of the results from a query operation, and it is also a class inherited by View. Query must be derived from a Table or a View. In addition, the stereotype Query has a Boolean type attribute named *isDistinct*. In the UML model, *isDistinct* is a tagged value placed in the head of a Query class with OCL expression. When the *isDistinct* value is true, all data records are unique in a data set. Conversely, a data set contains all of the data records resulting from a query operation.

3.2 Specification of the stereotype Column

Figure 3 shows a metamodel specifying a stereotype Column and its attributes. In the database, a column is owned by a table or a view used to represent an actual column or a virtual column. This class owns five attributes: kind, key, *isRequired*, table, and column. Attribute kind is a type of enumeration *ColumnKinds* with listing values: actual, derived, calculation, aggregation, subquery, function, and invariant. In a UML model, attributes are a tagged-value, adapted to a model element. Table 1 presents details of the specification related to the tag kind.

Attribute *key* is another attribute owned by the stereotype Column. This attribute is an instance of a type of enumeration *KeyKinds*, containing three values PK, FK

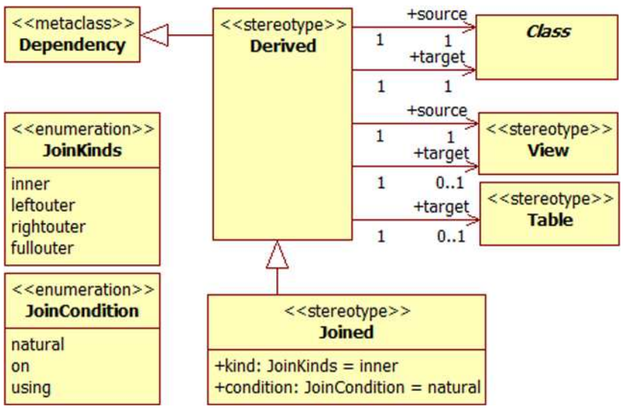


Fig. 4: Metamodel of Derived and Joined.

Table 2: List of tagged values for Joined's tag: *kind*

Value	Description
inner	This value explains that the relationship is an inner join calculation.
leftouter	This value shows that the dependency relationship is a left outer join calculation.
rightouter	This value indicates that the relationship is a right outer join relational calculation.
fullouter	This value denotes a relationship as a full outer join relational calculation.

and AK. These values are used to state a column which can be a primary key, a foreign key, or an alternative key. The Boolean attribute *isRequired* is used to indicate whether a column can have a null value or not. String type attributes table and column are used to fill the names of data source tables and columns.

3.3 Specification of stereotypes Derived and Jointed

Figure 4 illustrates stereotypes Derived and Joined. These two stereotypes represent dependency relationships in the UML model. Derived shows a relationship that provides a connection between a source class and a target class. The source class must be a View, and the target class can be a View or a Table. The stereotype Joined is also a dependency relationship, because it is inherited from a Derived relationship. In the database, a view must be derived from a table or an existing view. Therefore, in modeling, we draw Derived from a View to a Table or another View. In this profile, a Query is also a View class, and we also draw Derived from a Query to a Table or another View in a UML model.

Stereotypes denote a joining operation based on relational calculation of Cartesian products. In SQL, join operations take two relationships and return another relationship as the result. A joined operation must have

Table 3: List of tagged values for Joined's tag: *condition*

Value	Description
natural	This value represents a natural join condition in a Joined relationship.
on	This value indicates the one condition in a Joined relationship.
using	This value illustrates the using condition in a Joined relationship.

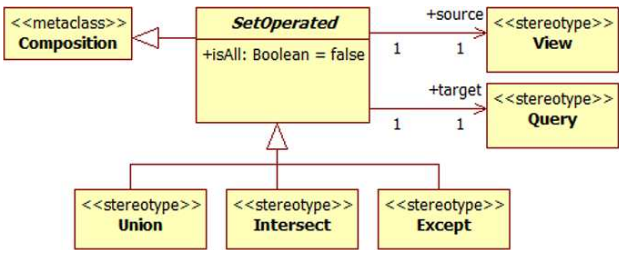


Fig. 5: Metamodel of SetOperated composition.

two parameters: type and condition. Table 2 shows an instance of enumeration a tagged value, kind. *JoinKinds* are applied to represent various join types in a Joined relationship. Another parameter in calculating a Joined relational is join condition. This approach defines a tagged value to represent a join condition. An attribute named condition in the Joined class is a *JoinCondition* type, indicating a join condition in a Joined relationship. Table 3 lists tagged values of join conditions.

3.4 Specification of stereotype SetOperated

As for the relational part, set operations include three kinds of relational algebra: union, intersection, and set difference. This paper defines an abstract stereotype *SetOperated* to indicate the set operations shown in Figure 5. It is a Composition relationship containing a Boolean attribute named *isAll*. In the relational calculation, a set operation automatically eliminates duplicates. If we want to retain all duplicates, we must write SQL using union all, intersect all, and except all. In modeling, we use *isAll* to denote whether all duplicates are retained. For relational set operations, this paper also defines three stereotypes *Union*, *Intersect*, and *Except* to draw relational algebra unions, intersections, and set difference calculations. A *SetOperated* relationship connects two classes, one of which is a source, and the other a target class. The source class must be a View and target class must be a Query.

3.5 Specification of stereotype Nested

Figure 6 shows a nested association metamodel, previously defined in our UML profile. It illustrates a

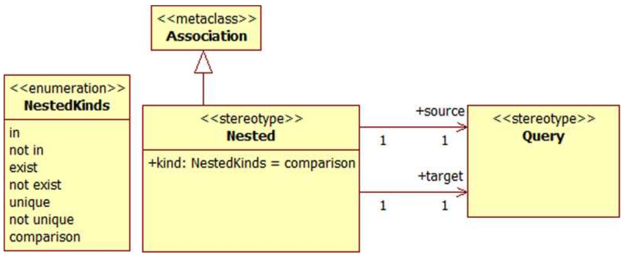
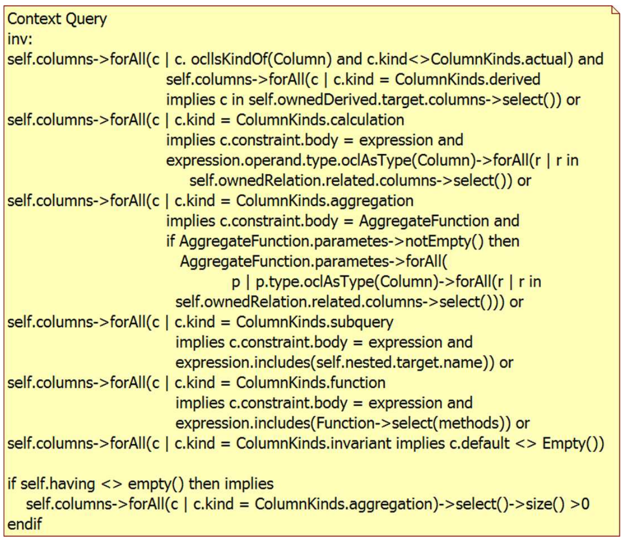


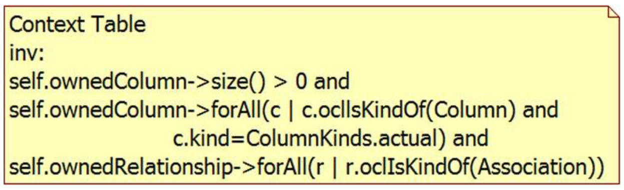
Fig. 6: Metamodel of Nested association.



```
Context Query
inv:
self.columns->forAll(c | c. oclIsKindOf(Column) and c.kind<>ColumnKinds.actual) and
                        self.columns->forAll(c | c.kind = ColumnKinds.derived
                        implies c in self.ownedDerived.target.columns->select()) or
self.columns->forAll(c | c.kind = ColumnKinds.calculation
                        implies c.constraint.body = expression and
                        expression.operand.type.oclAsType(Column)->forAll(r | r in
                           self.ownedRelation.related.columns->select()) or
self.columns->forAll(c | c.kind = ColumnKinds.aggregation
                        implies c.constraint.body = AggregateFunction and
                        if AggregateFunction.parametes->notEmpty() then
                          AggregateFunction.parametes->forAll(
                                p | p.type.oclAsType(Column)->forAll(r | r in
                         self.ownedRelation.related.columns->select())) or
self.columns->forAll(c | c.kind = ColumnKinds.subquery
                         implies c.constraint.body = expression and
                         expression.includes(self.nested.target.name)) or
self.columns->forAll(c | c.kind = ColumnKinds.function
                         implies c.constraint.body = expression and
                         expression.includes(Function->select(methods)) or
self.columns->forAll(c | c.kind = ColumnKinds.invariant implies c.default <> Empty())

if self.having <> empty() then implies
   self.columns->forAll(c | c.kind = ColumnKinds.aggregation)->select()->size() >0
endif
```

Fig. 7: OCL expression for Table class verification.



```
Context Table
inv:
self.ownedColumn->size() > 0 and
self.ownedColumn->forAll(c | c.oclIsKindOf(Column) and
                  c.kind=ColumnKinds.actual) and
self.ownedRelationship->forAll(r | r.oclIsKindOf(Association))
```

Fig. 8: OCL expression for View class verification.

stereotype *Nested* to explain nested subquery operations. Stereotype Nested is an association relationship, use to represent a subquery calculation, to test for set member-ship, make set comparisons, and determine set cardinality. This paper defines a tag kind to limit a *Nested* association for these calculations. *Kind* is an instance of *NestedKinds*. In a UML model, a *Nested* association can connect a Query to a Query class.

3.6 Verification rules

In this section, we present the verification rules declared by OCL to verify a model using our method, as MDD is based on correct models. Once the model has been designed, we can use several criteria to verify it in ensuring the accuracy of model transformation. Analyzing OCL invariants can reveal insightful information and ensure the correctness of the model transformation [4]. Therefore, this study declares a number of OCL invariants to specify verification rules for checking database models. We describe the verification rules and the implementation of OCL invariants. Due to limitations in the number of pages, we cannot list all of the OCL expression. Some of these are listed below:

–A *Table* class includes at least one Column attribute, and its own attribute tagged-value *kind* must equal *ColumnKinds.actual*. The relationship owned by a Table must be Association type. This verification rule in OCL expression is shown in Figure 7.

–A *View* class includes at least one attribute, and these attributes must be a type of Column. The tagged-value of all attributes must not equal *ColumnKinds.actual*. This class must have a relationship belonging to a type of Composition. Figure 8 shows an OCL expression used to express a rule for verifying a View class.

–A Query class includes at least one Column attribute, and the tag *kind* of this attribute must not equal *ColumnKinds.actual*. If the attributes of tagged-value *kind* equal *ColumnKinds.derived*, it must be derived from an attribute of a Table Column. Although the attributes of tagged-value *kind* equal *ColumnKinds.aggregation*, it is an aggregation column calculated by an aggregation function. Moreover, if the parameters of an aggregation function belong to Column type, they must exist in attributes of derived classes. If the attributes of tagged-value *kind* equal *ColumnKinds.subquery*, the attribute must be a Column constructed by a database query operation. A subquery Column must contain an OCL constraint expression, and this constraint should be a Query class name. If the attributes of tagged-value *kind* equal *ColumnKinds.function*, its OCL constraint expression must include the name and argument of the function. Although the attributes of tagged-value *kind* equal *ColumnKinds.invariant*, it is an invariant Column attribute, and must be assigned a value as a default. If this class contains an OCL expression and this expression includes a tag, it must check its owned attributes with at least one aggregation Column. The OCL expression is shown in Figure 9.

4 Case Study

In a UML model, a stereotype is represented as a string between a pair of guillemets (« ») or as a new icon. A tagged value specifies a new kind of property that it is attached to a model element and rendered as a string enclosed by a pair of braces ({ }). A constraint is a

```
Context View
inv:
self.ownedColumn->size() > 0 and
self.ownedColumn->forAll(c | c.oclIsKindOf(Column) and
                    C.kind<>ColumnKinds.actual) and
self.ownedRelationship->size() = 1 and
self.ownedRelationship->oclType() = oclIsKindOf(Composition)
```

Fig. 9: OCL expression for Query class verification.

Table 4: The schemas used for the ordering information system

Customer = {Id, Name, City, Address}
Product = {Id, Name, Category, Price}
Order = {OrderNo, OrderDate, CustomerId}
OrderItems = {OrderNo, ProductId, Quantity, Price}
Supplier = {Id, Name, City, Address}
SupplierProducts = {SupplierId, ProductId}

Table 5: Simple SQL code for the retrieval of information

SELECT O.OrderNo, O.ProductId, P.Name AS ProductName,
O.Quantity, O.Price, O.Quantity * O.Price AS Amount
FROM OrderItems AS O
INNER JOIN Product AS P ON (O.ProductId=P.Id)

specification for model elements, attached to any model element to refine its semantics, and can be defined by means of an informal explanation using Natural Language or by means of OCL expressions. It is also rendered as a string enclosed by a pair of braces ({ }). This section provides a case study to demonstrate the benefits of using our modeling approach for database information retrieval modeling. Table 4 shows the relation schemas used in the examples in this study, in an ordering enterprise. In this case, the enterprise is based on a scenario including customers, orders, and products. A customer can place one or more orders, and an order can be the purchase of more than one product.

4.1 A simple information retrieval model

A record of an ordered item is stored to two separated tables *OrderItems* and *Product*. The user requires all of the information related to the items in an order. With this requirement, we designed the UML model shown in Figure 10. This model contains two classes with «Table» named *OrderItems* and Product to state two tables, and also draws a class with «Query» to denote a query operation. The class *OrderItems_Product* is a query result, derived from *OrderItems*, and joined to *Product* with a condition *on (O.ProductId=P.Id)*. This join is an inner join type.

The primary goal of MDD is to model the transformation of code. Table 5 shows the SQL code transformed from a UML model, shown in Figure 10. For

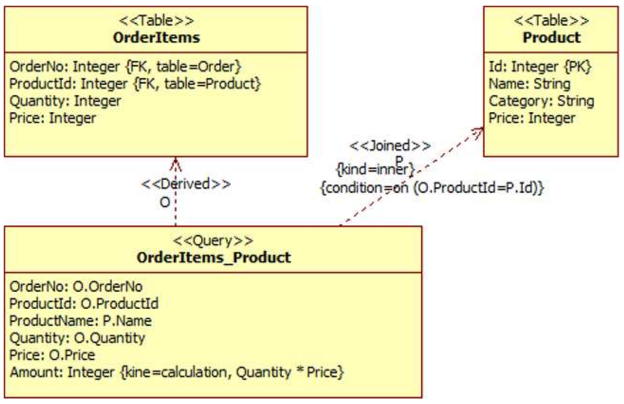


Fig. 10: A simple UML model for the retrieval of information.

this model transformation, the first step involves ensuring whether a *Query* class is not connected to the target end of a relationship. In this example, *OrderItems_Product* is not connected to a target end. The next step is transforming this class to an SQL select clause. The third step is to determine the *Derived* relationship from connections owned by the class, and transform the class name into SQL from the clause. Finally, we must determine the Joined relationship connected to this class, and transform the target class into an SQL join clause.

4.2 Set operations modeling

Set operations include three kinds of relational calculation: union, intersection, and set difference. In relational algebra, the relationship associated with operations must be compatible; that is, they must have the same set of attributes. Now, we want to retrieve all orders by quarters; however, order information is stored in a table with no *quarter* attribute. The table contains only the attributes such as *OrderNo*, *OrderDate*, and *CustomerId*. We can calculate *OrderDate* according to a function of date, such as YEAR() and MONTH(), built into the DBMS to separate records into four sets and then join these four sets into one. Figure 11 represents a union set operation models. In this case, we first draw a *Query* class and a *Derived* relationship to connect to the *Order* class, and then set the OCL expression to attach according to this relationship, to limit query operations. Next, we draw another *Query* to denote a new query operation, after which the *Query* class is completed. Finally, we draw a *Query* class and a composition relationship connecting those four *Query* classes. The SQL code used to transform this union operation model is shown in Table 6.

To model intersection, simply change the *Union* composition relationship to *Intersect*, and design a set difference model only to change the *Union* composition into an *Except* composition. Figure 12 shows an example

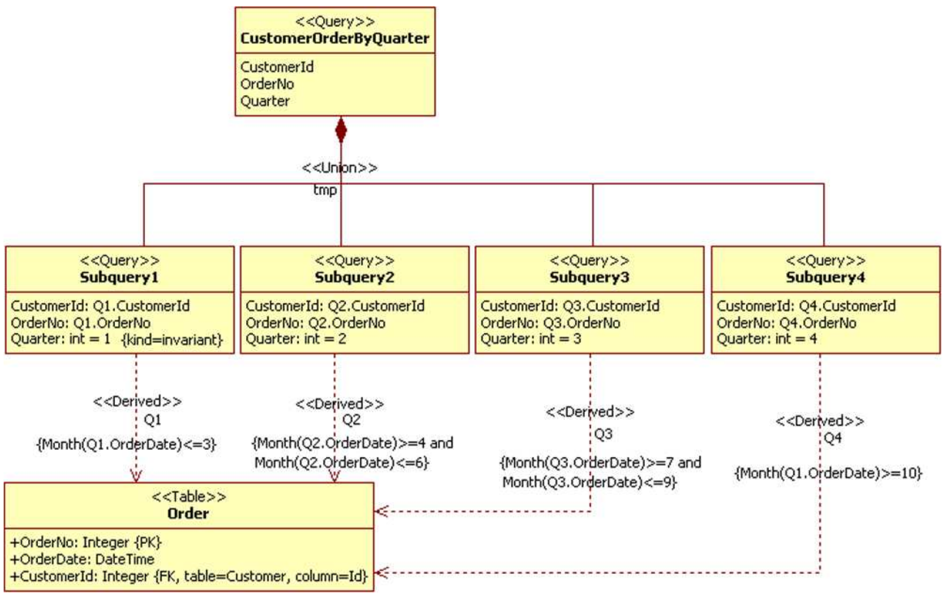


Fig. 11: Union set operation model.

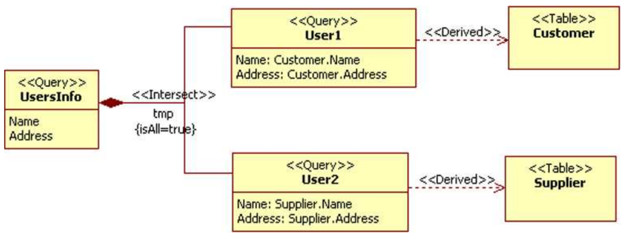


Fig. 12: Intersection set operation model.

used to illustrate an intersection operation. To identify all users (customers and a supplier) in the ordering system, we draw a composition with an attached tagged value of *isAll=false*. The intersect operation automatically eliminates duplicates. If we want to retain all duplicates, we must assign tag *isAll* to*true* value. The SQL code for an intersection set operation is stated in Table 7.

4.3 Nested subquery modeling

Subqueries are commonly used to perform tests for set membership, make set comparisons, and determine set cardinality. Figure 13 illustrates a nested subquery to model a test membership for the requirements in the retrieval of information. This example shows that to find all customers who have placed orders, we nest the subquery in an outer select. The resulting query is listed in Table 8.

Figure 14 illustrates another nested subquery to find users with both *Customer* and *Supplier* at which

Table 6: Union set operation SQL code

```
SELECT CustomerId, OrderNo, Quarter
FROM
(
    SELECT Q1.CustomerId, Q1.OrderNo, 1 AS Quarter
    FROM Order AS Q1
    WHERE MONTH(Q1.OrderDate) <= 3
    UNION
    SELECT Q2.CustomerId, Q2.OrderNo, 2 AS Quarter
    FROM Order AS Q2
    WHERE MONTH(Q2.OrderDate) >= 4 and
MONTH(Q2.OrderDate)<= 6
    UNION
    SELECT Q3.CustomerId, Q3.OrderNo, 3 AS Quarter
    FROM Order AS Q3
    WHERE MONTH(Q3.OrderDate) >= 7 and
MONTH(Q3.OrderDate) <= 9
    UNION
    SELECT Q4.CustomerId, Q4.OrderNo, 4 AS Quarter
    FROM Order AS Q4
    WHERE MONTH(Q4.OrderDate) >= 10
) tmp
```

Table 7: Intersection set operation SQL code

```
SELECT Name, Address
FROM (
    (SELECT Name, Address FROM Customer)
    INTERSECT ALL
    (SELECT Name, Address FROM Supplier)
) tmp
```

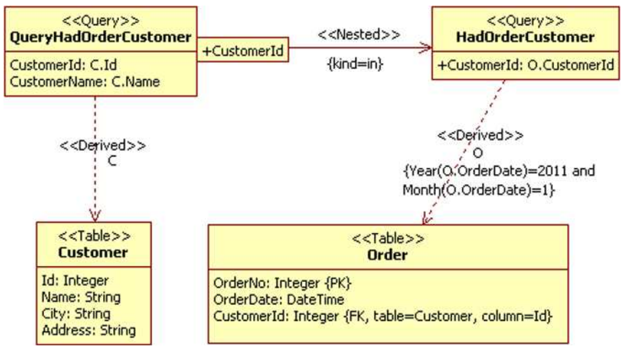


Fig. 13: Nested subquery operation - test membership.

Table 8: SQL code: Nested subquery operation - set membership

```
SELECT C.Id AS CustomerId, C.Name AS CustomerName
FROM Customer AS C
WHERE C.Id IN (
        SELECT O.CustomerId
        FROM Order AS O
        WHERE YEAR(O.OrderDate)=2011 and
MONTH(O.OrderDate)=1)
```

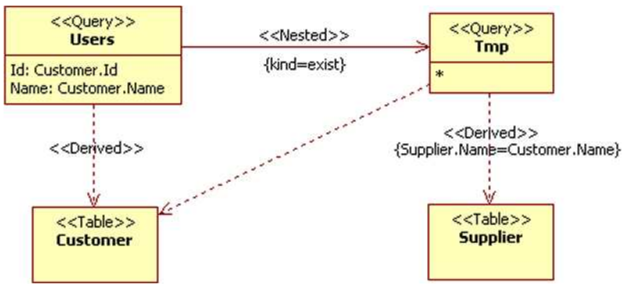


Fig. 14: Nested subquery operation - test empty relationships.

Supplier.Name equals *Customer.Name*. In the modeling, we draw a *Users* class and a *Nested* relationship with an attachment *{kind=exist}* to connect to *Subquery2* class. The *Users* class represents a query to retrieve data from *Customer*, connected to an outer select to test for empty relationships. *Subquery2* is an outer select that retrieves data from *Supplier* with a condition. To realize MDD, we must also transform the model to code. Table 9 lists the SQL code transformed from the example model to SQL code, as shown in Figure 14.

4.4 Aggregations modeling

In the database, aggregating operations involve calculus using an aggregate function. Most existing DBMS provide five built-in aggregate functions: *avg*, *min*, *max*, *sum*, and *count*. These five functions calculate average, minimum, maximum, sum, and count. An example aggregation model is shown in Figure 15. This example explains the requirements to find the count and sum of a customer order. In *CustomerOrder_Sum*, we add two attributes to denote aggregation. The attribute *OrderCount* is an aggregation column applied to count orders using a COUNT function with a parameter. Attribute *TotalAmount* is also an aggregation column,

Table 9: SQL code: Nested subquery operation - test for empty relationships

```
SELECT Id, Name
FROM Customer
WHERE exists (
      SELECT *
      FROM Supplier
      WHERE Supplier.Name = Customer.Name)
```

Table 10: Aggregation operation SQL code

```
SELECT T.CustomerId, COUNT(distinct T.OrderNo) AS
OrderCount,   SUM(V.Amount) AS TotalAmount
FROM Order AS T
INNER JOIN OrderItems_Product AS V
ON(T.OrderNo=V.OrderNo)
GROUP BY T.CustomerId
```

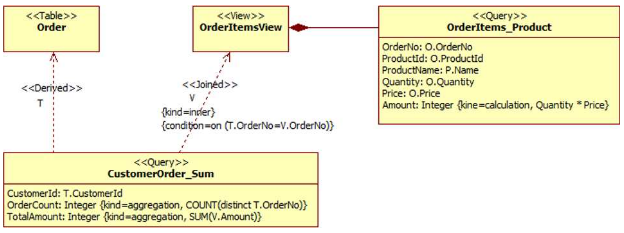


Fig. 15: Aggregating UML model.

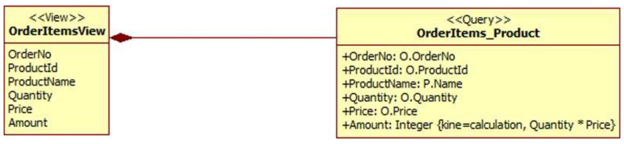


Fig. 16: A UML model of the database view.

Table 11: SQL code to define a view

```
CREATE VIEW OrderItemsView(OrderNo, ProductId,
ProductName, Quantity, Price, Amount)
AS
SELECT O.OrderNo, O.ProductId, P.Name AS ProductName,
O.Quantity, O.Price,
              O.Quantity * O.Price AS Amount
FROM OrderItems AS O
INNER JOIN Product AS P ON (O.ProductId=P.Id)
```

Table 12: Comparison summary of database modeling methods

Methods	Schema	Aggregation	Joined	Set operations	Nested subquery	Visual notation
ERD [16]	Yes	No	No	No	No	Yes
IE [7]	Yes	No	No	No	No	Yes
IDEF1X [8]	Yes	No	No	No	No	Yes
Ambler [2]	Yes	Yes	No	No	No	Yes
Gornik [10]	Yes	Yes	Yes	No	No	Yes
Balsters [3]	Yes	No	Yes	Yes	No	No
Torres et al. [19]	Yes	No	Yes	No	No	Yes
Song et al. [18]	No	Yes	Yes	Yes	Yes	Yes
Proposed Model	Yes	Yes	Yes	Yes	Yes	Yes

used to denote a sum calculus. We designed the model for transformation into SQL code. Table 10 shows a SQL code for an aggregating operation, transformed from this model.

4.5 View modeling

A view is the results from an information retrieval operation. It contains a number of attributes derived from tables or through calculus. In this paper, we defined a stereotype «View» to represent a view. Figure 16 shows a view model that is a composite by a *Query*. In the SQL, we define a view by using the *create view* command. Table 11 shows a SQL code to define a view, and this SQL code is transformed from the model shown in Figure 16.

5 Discussion and Conclusion

Table 12 lists a matrix, which compares various factors for a few different methods [2,3,7,8,10,16,18,19]. The factor "*Schema*" determined that a method can be used to represent a view schema in models. The "*Aggregation*" was used to indicate whether a method states aggregation in its models. The "*Joined*" was used to determine whether the method can denote join operations in its models. The "*Set operations*" was used to indicate whether a method can represent set operations such as union, intersect, and set difference in its models. The "*Nested subquery*" was used to indicate whether a method supports a nested subquery in its models. The "*Visual notation*" was used to determine which method provides visual notation.

The results of the preceding comparison demonstrate the superiority of the proposed modeling approach over existing methods. Our proposal illustrates complete information regarding the operations involved in information retrieval, while providing visual notation modeling language. We believe that this approach can provide the knowledge required to understand how data is organized in modeling a database. In recent years, studies in database modeling have proposed many approaches using UML. Most of these approaches use class diagrams to model relational database static schema. For modeling the dynamic operations of a database, a number of these approaches have used sequence diagrams or OCL in the design of the models. Nearly all of these approaches are useful for modeling the dynamic aspect or static schema of a database; however, none of them provide the means to represent information retrieval models using a UML class diagram.

In the application development phase, we designed a model that clearly and accurately captures the requirements of the user, using a visual modeling language capable of improving communication among end users and team members involved in the development of applications. Relational database query operations are based on set theory, and the schema of database views is based on query operations. Despite there are many relational database modeling approaches, most of them have only been applicable to the modeling of database table schema, and only a few of them deal with database views and query operations. However, in database modeling, we must consider complex columns and relationships of query operations. This paper proposes a UML profile to improve database modeling for information retrieval models. We believe that this approach provides the means to design information retrieval models capable of facilitating communication among application designers by providing a unified view for members of the development team and end users. This approach could also be beneficial in reaching goals in MDD and the automation of software system development.

Implementing such a highly formalized model in a practical application may have problems with the efficiency. However, the proposed database retrieval modeling method can be implemented into a computer-aided software engineering (CASE) tool, which enables the automatic generation of more comprehensive code in the MDD of enterprise applications. Therefore, it can save their development and maintenance effort. We have applied this modeling method with a tool support to several real cases of enterprise application development projects.

References

[1] M. Albert, J. Cabot, C. Gómez, V. Pelechano. Generating operation specifications from UML class diagrams: A model transformation approach. *Data & Knowledge Engineering*, **70**, 365-389 (2011).

[2] S. W. Ambler. *Agile Database Techniques*, Wiley Publishing, Inc. Canada, (2003).

[3] H. Balsters. Modelling Database Views with Derived Classes in the UML/OCL-Framework. *UML 2003 LNCS*, **2863**, 295-309 (2003).

[4] J. Cabot, R. Clarisó, E. Guerra, J. de Lara. Verification and validation of declarative model-to-model transformations through invariants. *The Journal of Systems and Software*, **83**, 283-302 (2010).

[5] B. Dobing, J. Parsons. How UML is used, *Communications of the ACM*, **49**, 109-113 (2006).

[6] L. Fakhar, A. G. Muhammad. Design of a simple and effective object-to-relational mapping technique. *Proceedings of the 2007 ACM symposium on Applied computing*, 1445-1449 (2007).

[7] C. Finkelstein, *An Introduction to Information Engineering: From Strategic Planning to Information Systems*, Addison-Wesley, Sydney, (1989).

[8] FIPS Publication 184. *Integration Definition for Information Modeling (IDEF1X)*. the Computer Systems Laboratory of the National Institute of Standards and Technology, December 21, (1993).

[9] L. Fuentes-Fernández and A. Vallecillo-Moreno. An Introduction to UML Profiles. *UPGRADE*, ATI, Barcelona, **V**, 6-13 (2004).

[10] D. Gornik. UML Data Modeling Profile. Available at: http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/tp162.pdf, (2003). (Accessed 11 July 2010)

[11] ISO/IEC 9075 Standard. *Information Technology - Database Language SQL:2003*. International Organization for Standardization, (2003).

[12] OMG. Model-Driven Architecture Guide Version 1.0.1. Available at: *http://www.omg.org/cgi-bin/doc?omg/03-06-01*, (2003). (Accessed 11 July 2010)

[13] OMG. Object Constraint Language Version 2.0. Available at: *http://www.omg.org/spec/OCL/2.0*, (2006). (Accessed 11 July 2010)

[14] OMG. Unified Modeling Language: Superstructure version 2.1.1. Available at: *http://www.omg.org/cgi-bin/doc?formal/07-02-05*, (2007). (Accessed 11 July 2010)

[15] OMG. Request For Proposal Information Management Metamodel(IMM). Available at: *http://www.omg.org/cgi-bin/doc?ab/05-12-02*, (2005). (Accessed 17 July 2011)

[16] P. P. Chen. The Entity-Relationship model: toward a unified view of data. *ACM Transactions on Database Systems*, **1**, 9-36 (1976).

[17] Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts 4th Edition*. The McGraw-Hill Companies, Inc., New York, (2002).

[18] E. Song, S. Yin, I. Ray. Using UML to model relational database operations. *Computer Standards & Interfaces*, **29**, 343-354 (2007).

[19] A. Torres, R. Galante, M.S. Pimenta. A synergistic model-driven approach for persistence modeling with UML. *Journal of Systems and Software*, **84**, 942-957 (2011).

[20] J. Zubcoff, J. Trujillo. A UML 2.0 profile to design Association Rule mining models in the multidimensional conceptual modeling of data warehouses. *Data & Knowledge Engineering*, **63**, 44-62 (2007).

[21] J. Zubcoff, J. Pardillo, J. Trujillo. A UML profile for the conceptual modelling of data-mining with time-series in data warehouses. *Information and Software Technology*, **51**, 977-992 (2009).

[22] J. Zhang, P. Feng, Z. Wu, D. Yu, K. Chen. Activity based CIM modeling and transformation for business process systems. *International Journal of Software Engineering and Knowledge Engineering*, **20**, 289-309 (2010).

[23] A. Armonas, L. Nemuraitė. Pattern Based Generation of Full-Fledged Relational Schemas from UML/OCL Model. *Information Technology and Control*, **35**, 27-33 (2006).

[24] Chaoyu Lin, Jyhjong Lin, Weipang Yang, An Architecture-Centered Method for Rapid Software Development, Applied Mathematics & Information Sciences, **6**, 479S-488S, (2012).

Chih-Min Lo received his Master degree in Information Management in 1996 from National Central University, Taoyuan, Taiwan, and the PhD degree from National Taiwan University of Science and Technology, Taipei, Taiwan, in 2012. His main research interests include software engineering, object-oriented technology, database management system and software information system development. He is currently an associate professor and director in the Department of Digital Media Design, Hwa Hsia Institute of Technology, New Taipei, Taiwan.

Hsiu-Yen Hung is a Lecturer of Digital Media Design at Hwa Hsia Institute of Technology, New Taipei, Taiwan. She is also a doctoral student in the Graduate Institute of Design Science in Tatung University, Taipei, Taiwan. She received the Master degree in "Architectural Design" at National Taiwan University of Science and Technology, Taipei, Taiwan. Her main research interests are: Web application systems, Web design, Human Engineering and 2D Animations.