

CHAPTER 2

LITERATURE REVIEW

2.0 OVERVIEW

This literature review is initially done on OOP, RDBMS, OODBMS and ORDBMS in terms of their basic concept and capability. Later on, their inter-relationship, pros and cons of each DBMS will be examined. Special attention will be given on how OOP relates to the three DBMS, or in other words, how the objects created in OOP application are stored in the DBMS. Finally, study will be performed to find out the conceptual theory or mapping pattern, previous research results, industries trend as well as the current implementation (such as any commercial tools or products) in mapping objects to RDBMS.

By following this sequence of literature review, it is expected to answer the questions as posted in section “Definition of problem” in Chapter 1. To the readers, this literature review will enable them to understand the terminology, design and implementation phase, and to support the intention, needs and contribution of doing this project.

2.1 OBJECT-ORIENTED PROGRAMMING (OOP)

The concept of object-oriented modeling is becoming increasingly practical because of its ability to thoroughly represent complex relationships, as well as to represent data and data processing in a consistent manner. This concept has been implemented on all facets of software and computer engineering; ranging from system analysis, system design, operating system, computer programming and database management system [5].

[1] In terms of OOP, object-oriented concept has changed the previously dominant way of doing programming, (i.e. procedural programming languages such as C, Cobol) that separate the data to be manipulated and function/routine that manipulate the data. In OOP, data (or **data value** in OOP term) and the operations that manipulate these data (i.e. **method** in OOP term) are encapsulated into objects. These objects can resemble the real world objects (such as person, car etc) as well as abstract concept (such as an event, account etc.). Objects interact with each other by sending messages to method of another objects in order to perform a specific task.

Besides **encapsulation**, fundamental concept on OOP that pertinent to this project (in terms of data in object) includes:-

1. **Inheritance** – the ability of an object to inherit the data value and method from another object. Its advantage is code reusability and a systematic way to group objects with similar attributes (data value) and behavior (method) into one **class**, which is a blueprint or template of objects. An object is an instance of a class. If class A inherits from another class B, then we say that class A is a **subclass** and class B is a **superclass**. This class inheritance can continue to many levels to form a **class hierarchy** [1].
2. **Composition or aggregation** – besides as mentioned earlier that object contains data, some objects may contains data and other objects (or a class may contains other class, i.e. **inner-class**) [5]. For instance, a person class may contain name, gender and Address class. Name and gender is primitive data type whereas Address class is considered to be a composite or **reference data type** of Person class.
3. **Association** – association is the relationship among multiple instances of class or objects [5]. Objects of a same class may interact with many objects of

another class and vice-versa. For instance, a student may take many courses and each course may be taken by many students. This is called as many-to-many (n:m) association. A special case of association is one-to-many (1:m), which is actually similar or hard to distinguish from composition [6].

With these advanced features of OOP, more and more programming language is migrating to OOP. Java is a pure object-oriented language from day one [2]. C has already migrated to C++ with object-oriented features. According to <http://msdn.microsoft.com/vbasic/productinfo/vbasic03/overview/>, the latest version of Microsoft Visual Basic.net has also become a truly object-oriented programming language that supports inheritance which formerly didn't.

2.2 RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS)

[7] Database management system (DBMS) has replaced the file system data management by having a pool of data that can be shared by multiple application programs and users concurrently. DBMS has eliminated the isolation of data at different location, data redundancy and provided ad hoc enquiry capability by using **structure query language (SQL)** – a universal non-procedural language that used to retrieve and manipulate data in database system efficiently and easily without having to write sophisticated computer program. DBMS also provide logical and physical data independence, so that changing of data structure or application program will not affect one another.

2.2.1 Fundamental Of RDBMS

For years, the dominant DBMS system is RDBMS due to its advantages over other database models, such as hierarchical database and network database model. The

relational database model is perceived by the user to be a collection of tables in which data are stored. These tables are link together by having a common field, which is the **primary key** (one or more fields or columns that uniquely identify each record or row in the table) and a **foreign key** in another table with the same data value as depict in the following schema:

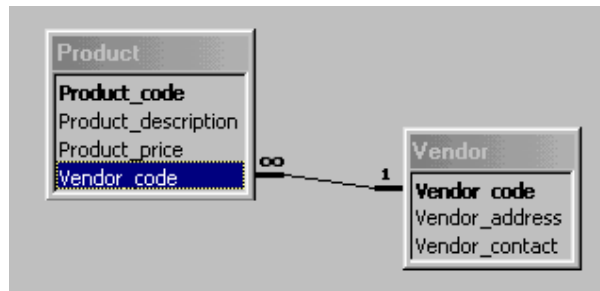


Fig. 2-1: Relational Schema With One-to-many Relationship

This relationship among tables can be one-to-one (1:1), one-to-many (1:M) or many-to-many (M:N) according to the business rules of the application. The latest version of SQL for RDBMS is SQL 2 and is now part of ANSI (American National Standard Institute) standard. Examples of RDBMS products are Oracle™ and Informix™.

2.3 OBJECT-ORIENTED DATABASE MANAGEMENT SYSTEM (OODBMS)

The object-oriented concept has been applied to DBMS and the result the relatively new DBMS called OODBMS.

2.3.1 Fundamental Of OODBMS

According to the author in article [f]: “*conventional RDBMS can no longer support the emerging complex database in real life. OODBMS or any other DBMS with*

object-oriented approach is urgently required to support efficiency data storage, retrieval and manipulation.”

[7] In contrast to RDBMS with tables that store only data but lack of manipulation capability, OODBMS contains objects with attributes or **instance variables** and **state**. Instance variables contain the data of the object which could be a simple data type or a reference to another object. Object state is the set of values that the object’s instance variables have at a given time. If we want to change the object’s state, we must change the values of the object’s instance variables by sending message to the object.

Each object is being uniquely assigned an identity, i.e. object ID (**OID**) which will never change throughout the life of the object. This OID is used to link different objects dynamically when objects interact with each other, in contrast to the permanent primary and foreign key relationship in RDBMS. Other object-oriented concept such as class, encapsulation of data and method, inheritance between subclass and superclass are also applied in OODBMS.

The following schema shows two objects with OID ObjX and ObjY which encapsulates 4 methods and the data. ObjX sends message to a method in ObjY to change or interrogate the state of ObjY.

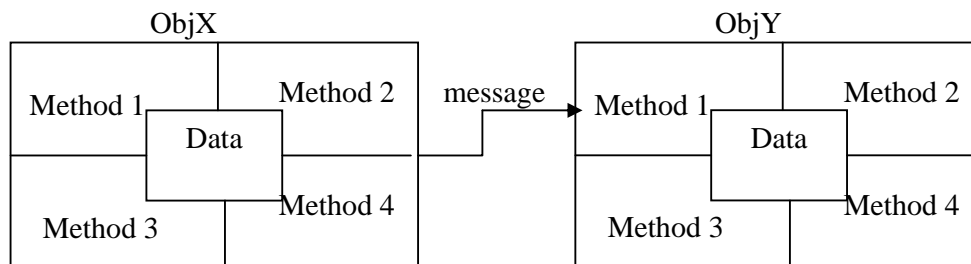


Fig. 2-2: Objects Send Message To Each Other

Examples of OODBMS products are Ardent™ developed by Ardent Software and ObjectStore™ developed by Object Design Inc.

2.3.2 Standard Organisation

[O] The **Object Management Group** (OMG) is an opened membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications. Some of the OMG specification are Model Driven Architecture (MDA), the Unified Modeling Language (UML), the Common Object Request Broker Architecture (CORBA), the Object Management Architecture (OMA) etc.

[P] While OMG is aiming at setting standard for interoperability of application under different platform, **Object Data Management Group** (ODMG) is the organisation that sets standard of storing objects which was not standardised initially. *“Before ODMG, the lack of a standard for object databases was a major limitation to their more widespread use. The success of relational database systems did not result simply from a higher level of data independence and a simpler data model than previous systems. Much of their success came from the standardization that they offer.”* Therefore, its primary goal is to put forward a set of standards allowing an OODBMS user to write portable applications i.e. applications that could run on more than one OODBMS. Eventually, this standards will be helpful in allowing interoperability between the OODBMS products as well, such as for heterogeneous distributed databases communicating through the OMG Object Request Broker.

[P] The ODMG standard consists of the object query language (OQL), object model, object defining language (ODL) and the bindings to OOP language (i.e. to

enable OQL to work closely with OOP language such as Java, C++ and Smalltalk). The latest standard is ODMG 3.0 that defines how to store objects in databases and design, develop, or implement object database products, or applications based on these products. The syntax for OQL is similar to SQL with some additional features such as object ID, complex objects, inheritance, polymorphism and relationships.

2.4 OBJECT/RELATIONAL DATABASE MANAGEMENT SYSTEM

[f] *“The main objective of object/relational database management system (ORDBMS) design was to achieve the benefits of both the relational and the object models such as scalability of RDBMS and support for rich data types of OODBMS. ORDBMS employs a data model that attempts to incorporate OO features in RDBMS.”*

[7] Another reason of this extension of RDBMS is because the RDBMS is no longer able to support the abstract data types such as graphic, multimedia data, engineering design and other complex scientific modeling. Hence, in response to the OO challenge, the functionality of conventional RDBMS has been extended to include the object-oriented features such as inheritance and support of complex data type, built on top of RDBMS that has been standardised and much easier to be understand and implemented by database designer. From an abstractive perspective; OODBMS is purely object-oriented like Java language which is *“object-oriented from day one”* as mentioned earlier. In contrast, ORDBMS is a hybrid of OODBMS and RDBMS, similar to C++ language (which is an extension from C, a procedural programming language) which supports both procedural programming and object-oriented programming.

[g] In ORDBMS, all database information is stored in tables, but some of the tabular entries may have richer data structure, termed abstract data type (ADT). ORDBMS supports an extended form of structure query language (SQL), called SQL3 in order to support ADT. American National Standard Institute (ANSI) and ISO (International Standard Association) SQL standardisation committees have, for some time, been adding features to the SQL specification to support object-oriented data management. This current version of SQL in progress has included these extensions. SQL3 object facilities primarily involve extensions to SQL's type facilities. Additional facilities include control structures to make SQL a computationally complete language for creating, managing, and querying persistent object-like data structures. The relational model has been drastically modified in order to support the fundamental features of OOP. Hence, the characteristic of ORDBMS are:-

- a. Base data type extension to support complex objects, i.e. ADT
- b. Support inheritance
- c. Allows user to define data type, functions and operators.

An example schema of a student table which ORDBMS supports is:

STUDENT (fname, lname, ID, sex, major, address, location, picture)

Note that the extra attributes “location” and “picture” which are not present in the traditional RDBMS. The data type of “location” is “geographic point” and that of “picture” is “image”.

[g] Many major existing RDBMS vendors are now extending or migrating their products into ORDBMS in respond to the challenge of OODBMS. IBM™, Informix™, Oracle™ and Sybase™ have already released object-oriented version of

their products. As a result there is now numerous vendors' support as compared to OODBMS.

2.5 COMPARISON OF THE THREE DBMS

We now look at why we are “going back to relational future” as stated by C.J. Date’s Third Manifesto [h] base on the comparison among the three DBMS and the pro and con of each DBMS.

According to article [g]:

“There is lack of standard, ad hoc enquiry and vendor support for OODBMS. ORDBMS as an extension to RDBMS can readily solve much of the problem and is expected to be dominant DBMS in 3 years.although OODBMS is implemented with pure object-oriented system and still evolving, but unable to live up to the expectation.”

And according to article [f]”

“The International Data Corporation (IDC) also expressed the opinion that the ORDBMS market will surpass the size of the OODBMS market in next 3 years.”

The table below gives comparison of the 3 database management systems [I] :-

Criteria	RDBMS	OODBMS	ORDBMS
Defining standard	SQL2	ODMG 3.0	SQL3
Support for object-oriented concept	Does not support. It is difficult to map program object to the database	Supports extensively	Limited support, mostly to new data types

Usability	Very easy to use	Satisfactory for OOP programmers but new to database designer without OOP knowledge	Easy to use except for some extension
Support for complex relationship	Does not support abstract data type	Supports a wide variety of data types and data with complex inter-relationship	Support abstract data types and complex relationships
Performance	Very good performance	Relatively low performance	Expected to perform very well
Product maturity	Relatively old and so very mature	This concept is few years old and so relatively mature	Still in development stage so immature
The use of SQL	Extensive support of SQL, i.e. SQL2	OQL (object query language) is similar to SQL, but with additional object-oriented features	SQL3, an extension of SQL2 (in its final stage), is being developed with OO features
Advantages	Its dependence on SQL, relatively simple query optimisation hence good performance	In can handle all types of complex application, reusability of code, less coding	Large storage capacity, massive scalability, high access speed and ability to query complex application and ability to handle large and complex application

Disadvantages	Inability to handle complex application	Low performance due to complex query optimisation, inability to support large-scale systems	Low performance in web applications
Support from vendors	Highly supported so the market size is very large but many vendors are moving towards ORDBMS	Presently lacking vendor support due to vast size of RDBMS market	All major RDBMS vendors are after this so has very good future.

Table 2-1: A Comparison Of Database Management System [I]

Besides, according to [f], *“OODBMS requires people to think differently, and in some cases relational users lack the OO foundation needed to work with OODBMS.... educating people on the OO foundation is a very painstaking process.”*

[7] Another disadvantage of OODBMS is that there need to be a way for the traditional system and OODBMS to communicate and work together. Consider about the legacy system that has been used for years and database application program written in Cobol language. This would take time, money and resources. Furthermore, the OQL doest not really support ad hoc enquiry as SQL does.

2.6 STORAGE OF OBJECTS CREATED IN OOP APPLICATION

This section will look into the storage of objects created in OOP application. We need to store the object because once the OOP application is terminated, all the objects data value will vanish from the main memory and not available for next use. Furthermore, latest multimedia data is in the form of object and the issue of storing object becomes more complicated and important [L].

Object persistence is an important issue in an OOP application. It describes the ways system allow data to be stored between execution of program. This persistence mechanism must support the movement of classes and objects between main memory and secondary/permanent storage [K].

[1] In Java 1.1 and above, objects can be stored directly to a binary file by using Java API class `ObjectOutputStream` and `ObjectInputStream`. However, as a nature of binary file, the data can only be used by the same application program and cannot be shared by other program without knowing the data structure of the binary file, i.e. the data has to be read in sequence and the data type has to be known in advance.

According to article [4]:

“If you want to use pure object-orientation to implement your business system you have to decide which database paradigm to use for your system. Today you have a choice of using:

- a. Object-oriented database systems (OODBMS) or*
- b. Object/relational access layers on top of a relational database”*

Article [J] also listed 3 object storage alternatives:

- a. File-based solution – which is only a satisfactory solution for simple storage requirements in most single-user applications.
- b. OODBMS-based solution – which provides full object-oriented support and full object instance support.
- c. RDBMS-based solution – which success is strongly based on the interoperability between OOP language and RDBMS.

Focus is now transferred to any existing tools or products that act as the interface between OOP and RDBMS for interoperability as stated in c) above. There is in fact some commercial software available as object-relational mapping tools or product, as those listed in [d] such as MTOS™, ObjectDRIVER™, ObjectBridge™ etc. There is also a book [e] that contains 225 detailed comparison tables on 13 object-relational mapping products.

Probably one of the products with very high prestige is Java Blend™, which is a product co-developed by Sun Microsystem, Inc. and Baan Company. The Java Blend development tool automatically translates and maps database records to objects and objects to databases. Java developers can concentrate on writing programs entirely in the Java programming language to retrieve and modify data from a database as objects. This increases programming productivity by eliminating the need to understand different database languages (SQL) or structural details – thus providing a simple, unified database programming model. *“This process is called object-relational mapping, and means that no additional programming is required”*[M]. However, this product is not included as part of Sun Java SDK API (which can be freely downloaded) and it is available only for sales without source code.

Besides, there is some related commercial software or works done by previous researchers as can be searched from Internet [b] [c]. However, all these programs or works are concentrate on analysing the class structure (act as a class browser) that helps the programmer to understand a given class file or source file that probably created by others. They are not a means for the programmers to reuse it in their program to store the objects' data value for later retrieval.

As a summary from above, 4 ways can be used to store objects:

1. To store the objects into a binary file (instead of DBMS) that allows the data to be retrieved by the same application program only. The semantic relationship among the objects can only be interpreted by the same application program.
2. To store the objects to OODBMS while maintaining the semantic relationship among the objects. However, the emerging OODBMS may not be readily available, lack of vendor support, relatively new in standardisation and interoperability, and will incur more cost and risk to the user by the change from RDBMS to OODBMS.
3. To store the object to RDBMS by implementing the object/relational mapping by the programmer himself. Hence, application programmers working on applications that access relational databases are forced to write massive conversion routines using two different languages, i.e. SQL and Java. They also have to concern about the database connectivity (such as JDBC or ODBC driver).
4. Similar to the third way as well, but instead of the Java programmer to create the mapping by his own, he makes use of a readily available tools or interface that can map the object to RDB and vice versa. All the mapping complexities are hidden by the mapping tools. The RDBMS seems to be an OODBMS to the programmer but in fact, the tools make use of currently most common RDBMS as a back-end to provide an “object-oriented” front-end/interface in storing the objects (or the data especially). It can be viewed as another form of ORDBMS because the tool has extended the functionality of the RDBMS with object-oriented features.

The forth resolution is the focus of this project. Hence, the next section will look into the basic concept of object/relational mapping, or the access layer between OOP application and RDBMS

2.7 OBJECT/RELATIONAL MAPPING AND ACCESS LAYER

This section contains the most important concept, i.e. the **object/relational mapping**, because the design of the tools in this project will be based on this foundation.

In order to create an interface or to fill in the gap between the two totally different architectures (OOP application that has a unique identifier or OID for each object that belong to certain class, whereas RDBMS uses primary and foreign key to define the relationship among tables), an **access layer** must exist in between as in figure below. Object-relational mapping is used to map Java or C++ objects to relational databases managed by Oracle, DB2, Sybase, and other RDBMS. Object-relational mapping is also known as OR mapping, O-R mapping, and O/R mapping [d].

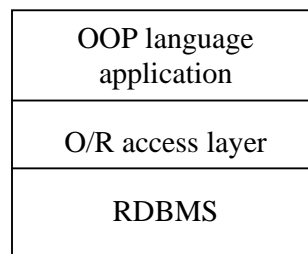


Fig. 2-3: Object-relational Access Layer

2.7.1 O/R Mapping Pattern [1] [6]

If the program is written in OOP language but has to use RDBMS instead of OODBMS due to some reason, an O/R mapping pattern has to be determined. In order to map all object models to RDBMS, the three basic OOP concepts (inheritance,

aggregation/composition and association as stated in Section 2.2) has to be considered as a minimum. Besides, the data types of OOP language and SQL data type in RDBMS has to be mapped as well (the discussion of data type mapping is delayed to later chapter because there are related to the OOP language and RDBMS being used. For instance, data type *int* (integer) in Java has to be mapped to data type *Number* in Microsoft Access™.)

There are some alternative mapping patterns to tackle each of the three OOP concept/relationship as listed in table below:-

Pattern for mapping composition	Pattern for mapping inheritance	Pattern for mapping association
1) Foreign key aggregation 2) Single table aggregation 3) Objects in binary large object (BLOB)	1) One inheritance tree one table 2) One class one table 3) One inheritance path one table 4) Objects in binary large object (BLOB)	1) Foreign key association 2) Association table 3) Objects in binary large object (BLOB)

Table 2-2: Alternative O/R Mapping Pattern [6]

BLOB is an acronym for binary large object. It refers to any random large block of bits that needs to be stored in a database, such as a picture or any binary file. The essential point is that a BLOB is an object that can't be interpreted within the database itself because it is a binary object [N]. An SQL BLOB is a built-in type that stores binary large object as a column value in a row of a database table. A BLOB object is valid for the duration of the transaction in which it was created [3].

In this project, we will use the highlighted patterns in the table above for each of the relationship, i.e. **“one class one table” pattern for mapping inheritance**, and

“foreign key” pattern for mapping composition and association. Other mapping patterns will not be discussed in this chapter.

2.7.2 “One Class One Table” Pattern For Mapping Inheritance

According to this pattern, to map classes in an inheritance hierarchy to relational database tables, one class (including abstract class) will be represented by one table. Consequently, the attribute of each class is mapped to individual table. An object ID (OID) is inserted into each table to link derived table’s rows with their parent table’s corresponding rows (Note: The focus of this project does not cover multiple inheritance which is not supported by Java.).

For example, with the inheritance hierarchical tree as below:

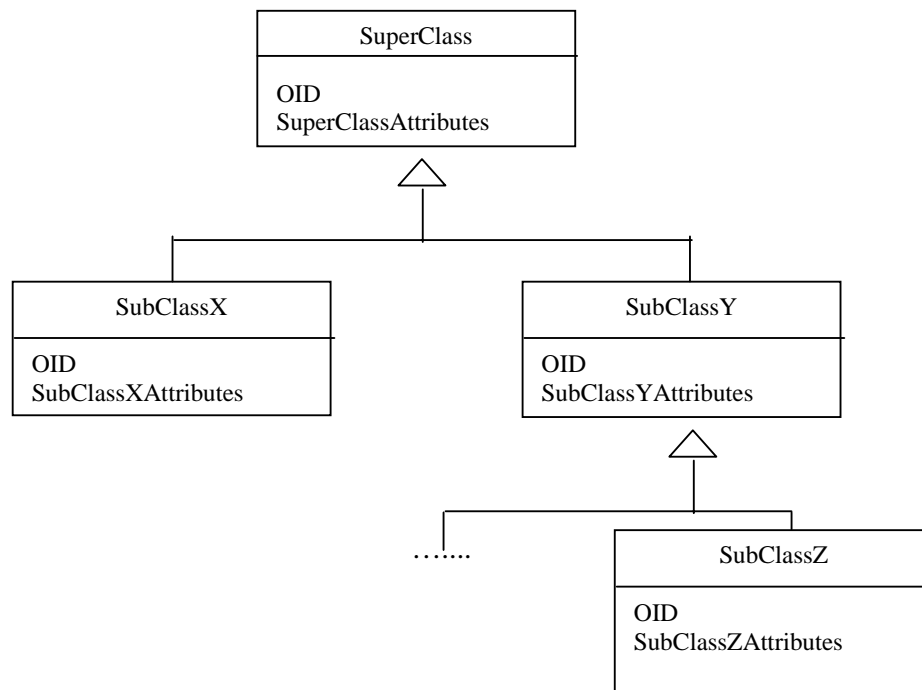


Fig. 2-4: Single Inheritance Hierarchical Tree

An instance of SubClassZ will be represented by 3 tables because there are 3 classes involved along the hierarchical path, i.e. the class of the object itself plus the 2 superclasses.

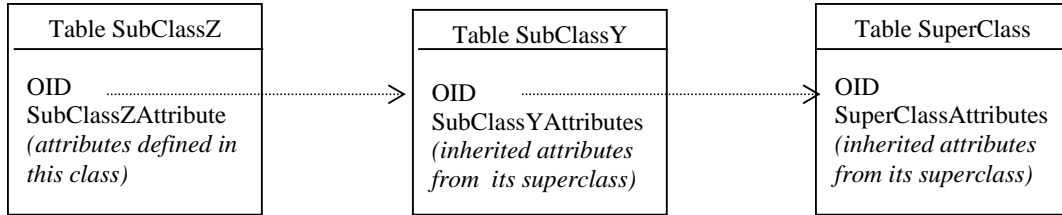


Fig. 2-5: Tables For Mapping Inheritance

2.7.3 “Foreign Key” Pattern For Mapping Composition And Association

In this project, we will limit our scope of work to “one-to-many” association, which is actually similar to composition as described in Chapter 2 Section 2.2.

Composition or association is the case where a class exists in another class (inner-class), or in other words a class member with reference data type (instead of primitive data type such as integer or character). This mapping pattern uses a separate table for the inner-class where the primary key of the “inner-table” is a foreign key of the table (or class) that contains the inner-class.

For example, a Person class contains a reference data type of address as below:

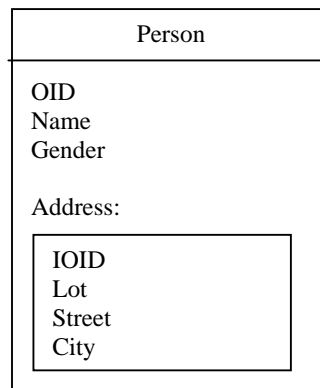


Fig. 2-6: A Class With Inner-class

For the Person class above, a synthetic object identity or **inner object identity (IOID)** is required in the table Person as foreign key which link to table Address.

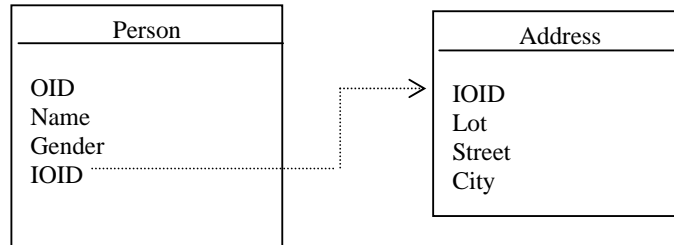


Fig. 2-7: Table To Map Inner-class

Note that the mapping pattern as described above is not the mapping algorithm yet. There are just the theoretical mapping strategies or blueprint that can be implemented. Mapping algorithm is the actual implementation or more precisely the coding on how to link the relationship among all tables and columns in the relational database, such as database structure, required tables and columns, tables and fields naming format, data type mapping, procedure of searching the superclass and inner class, OID generation etc. The structure of the database and table design has to be determined in order for this algorithm to work. This implementation must be determined based on the OOP language and RDBMS being used by considering the capability, limitation and performance. Therefore, discussion on the mapping algorithm will be delayed to later chapter and this works or solution is the main objective of this project as stated in Chapter 1.

2.8 CONCLUSION

As we have seen from previous findings and industrial trend, OOP will be the dominant programming language in probably next decade or even longer. Therefore, more and more application program will be written in OOP language and the storage

of data in the form of object will be the key issue of the success of application. Besides, the data being stored is becoming more complex, such as graphic and multimedia object. Hence, choosing the right DBMS to store the objects shall be carefully investigated while looking at their pros and cons, conveniences to the application programmers, flexibility, interoperability, cost incurred and the impact to the current system.

The object-oriented concept that swept through the computer world has now been challenged by the weaknesses, standardisation and lack of vendor support in OODBMS. The industry is now looking into developing a better DBMS on how to tackle current problem encountered in “pure object-oriented DBMS”. Much of the risk and difficulty may arise by moving directly into OODBMS which is completely new to the users, database designers and even programmers. The recent resolution and trend is to look into extending the RDBMS, whether by reconstruct a new RDBMS with object-oriented add-on features (i.e. ORDBMS), or to create an interface (object/relational access layer) between the OOP application and conventional RDBMS. The latter is the main focus and objective of this project.