Executive summary:

Many object oriented databases are available in market today, some of them are open source and some of them are commercial application. Today, available object oriented database provides almost all the facilities that are available in relational database and object oriented database has its own advantages over the relational database, even though relational databases are used more than the object oriented database. There are some reasons behind the less usage of object oriented database, but one of the major reasons is the, less functionality for the schema evolution. So this project is proposed to remove that drawback of the object oriented database.

The main reason behind less support for the schema evolution functionality in current available object oriented database is that they stores object persistently to permanent memory and due to this there is no isolation between stored object and its class. This project proposed new technique for the object oriented database which can provide partial isolation to stored object from its class so that new schema evolution technique can be achieved for the object oriented database.

- As a part of this project I have created new architecture for the object oriented database (see section 5 – My Design)
- I have provide some query tool and the refactoring (schema evolution) tool for my proposal (see Section 5.1.3 – Tools)
- I have compared my proposed object oriented database with other available database (See Section 6 – Result).
- I have proposed some suggestion which can further improve my technique of object oriented database (See section 7 – Future work)

Acknowledgement:

This dissertation would have not been possible without support of many people. I would like to thank all of them one by one. First of all I would like to thank, my supervisor Dr. Ian Holyer for his help, support and guidance throughout my project. I also want to thank Jatin, and Sarah who supports me to improve my report standard. Last but most importantly I want to thank my Family and Friends for their support throughout my masters.

Table of Content

1. Introduction:

Object oriented database stores object (instance of class which is basic of object oriented programming) into storage system and read it whenever required. It also provides functionality to update and delete objects from the storage. Today too many object oriented databases are available in market and almost all of them provides schema evolution function as automated technique but they are not so efficient when refactoring is done on large scale in the project [19][20][23][32][44]. By considering above facts, this project is aiming to overcome drawbacks of exiting object oriented databases by proposing new design and some tools which can handle above problem.

To make my project more understandable I will be describing some background information of my project which is covered in the background section of this dissertation. In background section I have explained topic like object oriented methodology, object oriented programming, database systems which includes relational database and object oriented database, Agile programming, object relational mapping, and schema evaluation. After background reading I have discussed state of the art in object oriented databases, this section gives information about currently available object oriented databases and point out their benefits and loose points. After the state of the arts section, problem statement is explained which point out the current problem in object oriented database, why it is needed to work on that and how I have overcome it in my proposal. In the next section I have discussed about the design I proposed for object oriented database which includes basic structure, tools that are developed for managing database, challenges that I have faced during my demo implementation, limitation of my developed demo using my proposed design, and proposal to improve my demo implementation. Next section after my design is results section which includes the advantages of my proposed architecture of object oriented database over current available object oriented database and some comparison with them. Finally this report discussed about some future work.

1.1 Aim and Objective:

Aim of this project is, to change the way how object oriented databases works, to provide complete schema evolution functionality without losing current advantages of object oriented database.

2. Background

This section presents background information to make my project title “Refactoring Object Oriented Database 2” more understandable. This includes topics which are essential to understand project title and my proposal of the object oriented database.

As this project titled “Refactoring object oriented database” it needs to understand the methodology of the object orientation and how this methodology works in the programming language. After that need to understand what is database and why it is require? Understanding of the available typed of database, detail information about the object oriented database and comparison of object oriented database with other database is also essential. These all detail helps to understand my approach for object oriented database. As this project is about refactoring object oriented database it is essential to understand what is refactoring in software engineering

and what are general steps to do refactoring in database? In this report new proposed architecture of object oriented database is inspired from the object relational mapping so knowledge of the object relational mapping and its advantages is essential.

According to above discussion following are some topics which need to know to understand my project title and my project proposal.

2.1 Object oriented methodology:

As project title is "Refactoring Object Oriented Database", to understand the methodology of the object orientation is essential and it will also be useful in understanding the object oriented programming languages and the object oriented databases, which are discussed below.

To facilitate reuse of software component into new system development, developers approaches the object oriented methodology. Using object oriented methodology system can use/share feature of/to other system [41].Use of the object oriented methodology leads to the high performance of the system, lower cost for their maintenance and high quality of the system [41]. Usually object oriented methodology applies to the single object model which is developed from the analysis and design phase of the system and it's carried throughout programming level [41]. This object in object oriented methodology contains data and the functions operating those data.

Object oriented methodology is based on following principals. The description is in general term so entity will represent the object.

Software development methodology is object oriented if it utilizes below listed four principals [3] [2]

- Abstraction of data and function principal: This principal means that computation is done individually in each entity. In short whole entity can work as one unit [2]. "This is a corner-stone principal of object oriented methodology" [2].

- Information encapsulation principal: encapsulation means information and implementation of mechanics of one entity is hidden from other entity. Reason for this point to shield information of the entity [2].

- Inheritance Principal: Idea of inheritance is to create new entity from existing entities but behavior of that entity is different than exiting one. And this can be done using child entity and parent entity where child entity is based on parent entity [2].

- Polymorphism for method: The idea behind polymorphism is ability to create same methods but functionality is related to specific entity [2]. This means that if X and Y class have same method so z can define polymorphic methods but functionality of that methods is different according to caller of X's entity and Y's entity [2].

By providing above four principal, object oriented methodology has following advantages [41]

- This methodology provides reusability of the developed system's component which improves productivity of new system.
- Above principals of the object oriented methodology provides high quality system.

- Using object oriented methodology system can detect error easily and can change functionality easily, which leads to the lower maintenance cost of the system.
- Object oriented methodology approach facilitate reuse of the components.
- Object oriented methodology also reduces the management complexity.

Above principals and advantages of the object oriented methodology makes it useful for medium to large scale projects. To develop the application using object oriented methodology many object oriented programming languages are available in the market such as Java, C#, C++, Objective C. As I am using java for my demo implementation of the object oriented database following are some information about java which can be useful to understand explanation of the demo implementation and new proposals for object oriented database.

2.2 Basic class and object Structure in Java

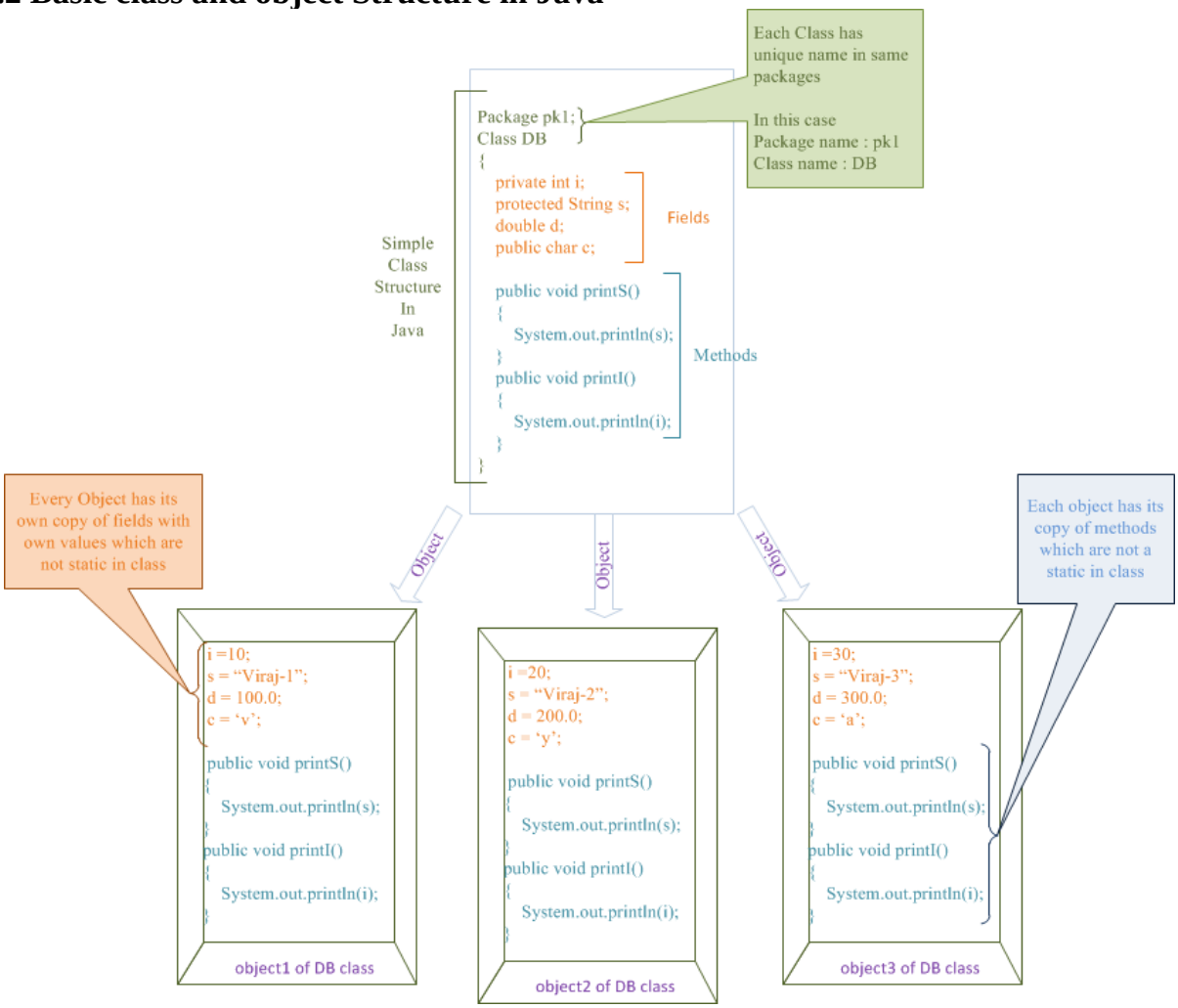


Figure 1 [Basic Structure of Java Programming Language]

Above figure shows basic structure of class and object in java. As shown in figure each class can contain fields (explained below) and methods (explained below). Object in software is quite similar to the real-world objects [14].Both contains states and its related behaviors. In software, object contains its states in fields and it expresses its behavior through methods [14]. Each object

contains copy of non static methods and the fields which are not static in class declaration in java [11]. Each object has different value for its copy of fields declared in class.

As shown in figure 1 the class is named DB. It has fields (variables) named i, s, d, c. First of all it is needed to understand what is field and types of variables.

Java contains four kinds of the variables.

- Instance Variable: This variable is called as instance variable because there value is unique to the class objects [27]. It means that all objects have different copy of the instance variable. This instance variable is also called as non-static field of class [27].

- Class Variable: This is called as class variable because only one copy created for class and each instance of class (object) uses the same copy. This variable is also called as a static Field of class [27].

- Local Variable: Same as object each method stores there local variables during method call. Variable declared inside the method is called as local variables [27].

- Parameters: variables that are passed as argument of the method are called as parameters of method [27] i.e. in function: public void testFunction(String str, int i); str and i are parameters of testFunction method.

As per the above discussion only instance and class variable are called as field in the java, so in explanation of my design I have mentioned about the fields which indicate these two variable types.

As per our discussion, the variables i, s, d, c in figure are called as the fields of the class. Now each variable has its variable type. i variable has data type int where "int" is a primitive data type [11]. s has data type "String". d has data type double. And c has data type char. If instance variable and class variables are refer as fields as per above discussion then variable data type for those type of variables is refer as the field type [11]. I have used field type in explanation of my design. So that field type is refers as data type of instance and class type variables in class.

Field type includes primitive data types, object (reference) data types, array of data, collection of data types and Enum type. Following some brief introduction about all field types [11] .

Primitive data type: eight primitive data type available for the java programming language. And they are byte, short, int, long, float, double, boolean, char. Apart from this primitive object java language has special support for the character String via class named "java.lang.String" [28]. All primitive type has default value if it is not declared in class.

Object (reference) data type: Any object of class is called as object (reference) type. All object has default value as "null" [28].

Array of data type: field can be array type. Array contains more than one value for same primitive or reference data type. Size of array can be declared at the time of initialization of array type [11].Java language also contains collection API. This API is useful to create array without specify their length [11].

Enum Types (Custom data Types): Enum type field contains fix set of constraints. All constrains are declared in enum. Enum can be defined by typing enum keyword instead of class [29].

Each object of class has fields of class and it has some value of that field. In above figure object1, object2 and object3 is shown. And each object has the field with its own value. For example object1 has fields i, s, d, c and all field has value "i" field has value 10, "s" field has value "Viraj-1" and so on. Each field can held value according to their field type. In case of object1 s has vales "Viraj-1" because it is of string type and "i" has value 10 because it is of int (Integer) type. As you can see all object contains different value for fields in different object.

As I show in figure each class may have methods and all instance of that class has one copy of the method if method is not static. If method is static you only one copy of methods per class is created.

Following are basic method structure.

```
Public double sumParameters (int value1, int value2)
{
        //calculation goes here.
}
```

Where public is access modifiers, double is return type, "sumParameters" is the method name.

int value1, int value2 are the parameters (as discussed above).

In the method return type and name is mandatory. You also need to mention a pair of parentheses, (), and a body between braces, {} [30]. Method understanding will help to understand my discussion.

My approach towards object oriented database is stores field value of object to my proposed files system instead of the direct (persistent) object storage. And to understand all terminology in proposed design explanation, above discussion can be helpful. By understanding above discussion it is easy to understand what are the schema changes inside the class (mentioned in section 2.7)?

2.3 Database:

In order to understand object oriented database it is needed to understand what is database, some concept of them, what are their types, what is object oriented database and how it works?

So Firstly, I made some introduction of the database and then type of database and their information in brief. At last I have explained about object oriented database and their advantages, disadvantages and improvement that can be made to improve it.

2.3.1 Introduction of database:

In Early 1970 when Computer was newly launched Files were used for Database. Before Database Management System(DBMS) came along, file processing system was used as an database [1].There are some problems with file processing system and they are data redundancy,

data inconsistency, difficulty in accessing data, data isolation, integrity problem, atomicity problem, concurrent access anomalies and some security problems[1]. To overcome problem of such database systems, Database Management System came into market with special kind of functionality to manage database systems. At the beginning Database usage was limited but after invention of internet in 1990, Database Management System got new direction, where any user can access global database [1]. Today database management system plays important role in Software Development. Some of big commercial applications are very well managed by DBMS. Today web applications, mobile applications and desktop applications are using DMBS widely, so performance of database is becoming a main concern. To do that in 1970 Dr. E. F. Codd in 1970 projected the relational database [10]. After that too many database come to the market.

Databases are used to store data in proper format and fetch data as it requires, delete un-useful data, and modify stored data. Today many types of database available in the market like relational database object oriented database and graph oriented database, some of them are commercial database application and some of them are open source database applications.

When you are using object oriented language and relational database management system, this tends to create object-relational mismatch problem [31]. This problem resulted in to development of object relational mapping. Today some good object relational mappings frameworks are available in market i.e. Hibernate, top link. This approach leads towards drain in system performance and increase the software complexity [20]. And this approach doesn't lead towards distributed and zero administrator architecture those required for embedded devices, Mobiles Applications, real-time systems [20]. One way to solve this problem is to give object's direct relationship with database and Object Oriented Database come up with same idea. This Object oriented database approach leads towards zero administrator architecture. This approach reduces the development time because, no administration is required for database, no database schema architecture required. Today for different object oriented languages different Object oriented databases are available in the market. DB4O is for Java and it is most popular and advance object oriented database. This OO database has very wide user community.

Relational database type is used widely today because of their availability, reliability, security and standardization etc. Object oriented database almost have same features like relational database and it has some advantages over the relational database, if user is using object oriented programming language to develop application, but because of some drawback of object oriented database make them less reliable (explained in section 3.5). This is the reason for less growth of object oriented database compare to the relational database [6].

One of the main reasons who take reliability from the object oriented database is that they are not good dealing with the class structure change (for more information section 4). My project is focusing on this issue, and tries to overcome this issue by proposing new design of the object oriented database (explained in section 5).

Currently available databases support some common functionality like transaction, and constraints. Before going through the database type, let understand some basic about what is transaction and what are the properties of the transaction? This is useful to understand how my proposal can support transactions.

Following explanation is the general properties of transaction that are needed in every type of the database.

2.3.2 Transaction in Database

Transaction is a set of programs execution that possibly updates various data items. In sort transaction is set of operations. Database should follow four properties to support transaction in proper way. Sometimes this property is called as ACID property of database transaction and they are [1]

- Atomicity: Either all operations of transaction have to done or none of them. This means that if some conflict found in middle, the transaction should have capability to rollback to initial level. [1]
- Consistency: This property shows transaction should be in isolation, so that it prevents consistency of the database. [1]
- Isolation: when multiple transactions are done in system then system must guarantee that one transaction must finish work before other starts. [1]
- Durability: after a transaction finished the changes it made, must persist over the system failure. [1]

Above four properties are the transactions properties require to implement in all type of the database to make them reliable. Following, in the explanation part of the object oriented database I have explained how transaction works for the object oriented database, which helps to understand how my approach of object oriented database can support same functionality.

As I have proposed constrains in my design (new proposal) of the object oriented database, some of them are not supported by the currently available object oriented database [20] [22] [32] for java programming language. So in this report I have explained all the type of constraints inside the explanation of the relational database because relational database supports almost all constraints.

As I told earlier, today many types of database are available in the market and most widely used database is relational database. So first see what the relational database is and what are their advantages? After that I have explained what object oriented database is and discuss about their advantages and disadvantages and some comparison with the relational database.

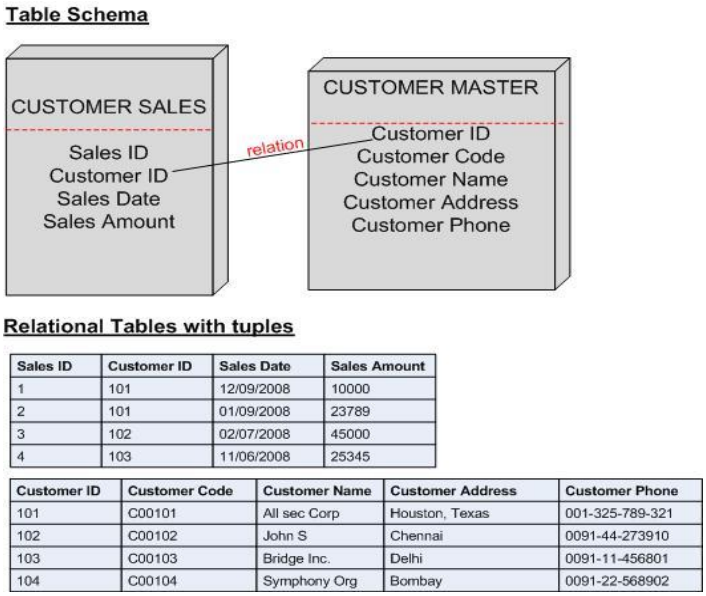
2.3.3 Relational Database:

A relational database uses relational models. Relational data model is developed by Dr. D. F. Codd who worked as IBM researcher. In 1985 he published rules for ideal relational database [10]. Following are some rules [10]

- All data should be offered as table form.
- All data should be accessed individually through some uniqueness.
- Some field can remain empty and empty field should contain null value for string represented data and 0 for number representation.
- All relational databases have access to the structure of its table.

- The database must have clearly defined Database languages i.e. SQL
- The database has to support views, where view is logical combination of data-field of different table. And views can perform data manipulation language in database same as table.
- User can get, insert or update multiple rows of same table or different table instead of one row of one table.
- Database has completely isolation from system method of storage. In short insert and update methods are not depend on system architecture.
- Database should support constraints on user input and to maintain its integrity
- Users of database system are not able to find, whether database is distinguish or not.
- Any user is not able to change or retrieve data without Database language query.

Any databases which satisfy the above rules are good relational database. In relational database, data are stored in tables and table contains rows and columns, where row is set of data for table which is called as tuple. Column is same data-typed set of data for particular tuple which is called as Attribute. In relational database table can be related to another table through foreign key. Most relational database uses Structured Query Language (SQL) as their database language [1].



Sales ID	Customer ID	Sales Date	Sales Amount
1	101	12/09/2008	10000
2	101	01/09/2008	23789
3	102	02/07/2008	45000
4	103	11/06/2008	25345

Customer ID	Customer Code	Customer Name	Customer Address	Customer Phone
101	C00101	All sec Corp	Houston, Texas	001-325-789-321
102	C00102	John S	Chennai	0091-44-273910
103	C00103	Bridge Inc.	Delhi	0091-11-456801
104	C00104	Symphony Org	Bombay	0091-22-568902

Figure 2 [Relational database schema and table structure (copied from [35])]

Figure 2 shows the basic structure of the relational database. As shown in figure 2, two tables are represented in schema form and in table tuple form and their name are customer sale and

customer masters. Both of them contains some colums (fields). As you can see in figure 2, data are stored inside the tables and each column (Attribute) contains the same type of data. Figure 2 shows the relation between two table colums. This relationship can be achived by foreign key and primary key constraints [1] [10]. Some of the constrains are dicuss below.

As I have proposed plan to provide constraint for my object oriented database, following topic will discuss about the constraints in relational database. By understanding this, reader can understand my proposal for constraint implementation (inside newly design object oriented database).

2.3.3.1 Constraints in Relational database:

Constraint is useful to restrict the domain of the attribute (column). Relational database provides different kind of constraints, and they are check constraints, Not Null constraints, unique constraints, primary key constraints and foreign key constraints [1] [10] . Support of different constraints varies in different databases. Above all constrains supported by oracle, which one of the major vendor of relational database. Following are some detail about the constraints in database.

Unique constraints: This constraint checks the unique entry for the column which is unique constrained [10].

Primary key constraints: This constraint is same as the unique constraints but this constraint does not allow null entry for the column (attribute) [10].

Foreign key constraints: This constraint is useful to give relationship between different columns. Generally a foreign key constrained column refers to the primary key constrained column. In figure 2, you can see that customer ID in customer sale table refers to the customer ID of the customer master table. As you can see in figure 2's table representation, all values of customer ID inside the customer sale is coming from the customer ID of the customers master table. So, foreign key constrained column's values comes from (related to) the referenced column's values of other table or same table [10].

I have proposed method to achieve above explained constraint inside new architecture for the object oriented database.

Today number of relational database system (RDBMS) available in market and major RDBMS are Oracle, SQL server 2005, My-SQL, Derby etc. Today available relational databases provide good security features and tools to handle relational database queries. These reasons show why relational database are used widely in industry. Today relational databases are used by all major programming languages, which also include object oriented programming languages.

Now let look at what is object oriented database, how it works, what are their advantages and what kind of drawback it has?

2.3.4 Object Oriented Database:

Object oriented database stores object directly to memory instead of storing its fields' values in table structure (different from relational database, and object relational mapping). This section shows brief information about the object oriented database and its manifesto, and after that it shows brief information about, how object oriented database works.

2.3.4.1 Manifesto of object oriented database

Object oriented database is on base of object based data model [1]. As we discuss above Dr. Codd propose rules for relational database, at that time object oriented concept was unstructured. There were no standard architectures for Object oriented database at that time [7]. In 1992 Malcolm Atkinson and five more people introduce one research paper named 'The object oriented database system manifesto' and this paper introduce some manifesto for object oriented database. This paper introduced some mandatory, optional and open choices features for object oriented database. This document discusses about some mandatory manifesto of object oriented database and they are as follow [7] [8] [5]

- **Complex Object:** All Object Oriented database should support complex object. Complex object means arrays, and list. They must have to provide appropriate operator to deal with such objects [7].

- **Object Identity:** In object oriented programming language every object has unique and immutable object identifier. Supporting object identity means OODB offers operation such as object assignment, test for object identity and object equality [7].

- **Encapsulation:** according to this manifesto object oriented programming language have to follow encapsulation and proper encapsulation obtain in OODB when only operations are visible but data and implementation of an operation is hidden inside the object [7].

- **Type of Classes:** this section consists of two part, data type and object class. In object oriented every class has a type and contains objects. And every object has value and value is described by types. Object oriented database should support this type of structure [7].

- **Type of hierarchies:** this point is indicating Inheritance. Child class automatically belongs to super class. Through inheritance attribute and methods are inherited from parent class/ super class. All Object oriented database has to deal with this feature properly.

- **Overriding, overloading and late-binding:** Overriding is to redefine method in subtype. Overloading mans same method with different versions. Late-binding is also known as virtual method dispatching [7].

- **Extensibility:** database has some predefine data type. And developer can define new type according to requirements [7].

- **Persistence:** Object oriented database system should have persistent storage concept to store object in memory. Persistent object means object will not destroy after termination of the project [7].

- **Secondary Storage management:** this is the classic feature of any database management system. This includes index management, data clustering, data buffering, query optimization and access path selection. Object oriented database should include this [7].

- **Concurrency:** While multiple users accessing same database concurrently then database system has to avoid concurrency problems.

- **Recovery:** Database system must be design in a way that, in terms of hardware or software failure, database should recover [7].
- **Ad Hoc Query Facility:** database should support high level query languages [7].

There are some optional features proposed by that paper. By implementing optional features object oriented database can become more powerful but they all are not mandatory features to make object oriented database. Some optional features are described below. [7]

- **Multiple inheritances:** to make object database more powerful it should support this feature. It is hard to deal with conflict resolution problem while implementing this feature. There are some rules available to deal with this kind of problems.
- **Distribution:** This characteristic is orthogonal to object oriented nature and that is why this feature is optional but this facility makes your database more usable.
- **Design transaction:** This is option feature but to make database more popular it must have to support business transaction robustly.
- **Versions:** This feature is also optional but, this feature give usage enhancement to object oriented database.

Some open choice manifestos are proposed in the paper [7] and they are "Programming paradigm, Representation System, Type System, Uniformity" [7]. Today available most object oriented database provides above all compulsory features and some of optional feature that above mentioned. Following detail shows some brief information how object oriented database works.

2.3.4.2 How Object Oriented Database works:

Object oriented programming languages store persistent objects in the temporary memory of the system [48]. User will lose the data if they restart the system or program. To avoid this java provides synchronized object storage in the file, but this method has some limitation [48]. To remove disadvantages of java synchronized object storage, object oriented database come up with the idea to store data into the memory from the temporary memory, and to do that user only need to call one function. Object oriented database also provides different functions to deal with stored objects in to database.

Following figure shows how the object oriented database works.

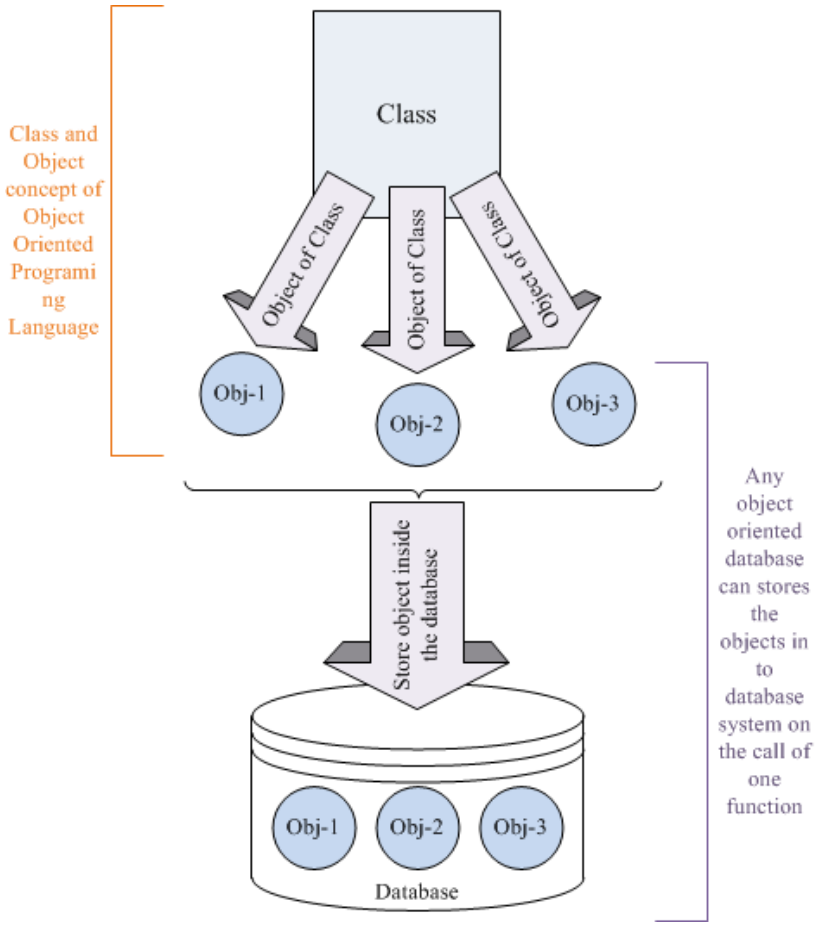


Figure 3 [Object Oriented database Basic Structure]

Figure 3, shows the basic structure of the object oriented database. As shown in figure object oriented language every class can creates their instance, which is called as the object. Object oriented database stores that objects inside the database and reads object from the database [48]. This database might be file system. Generally object oriented database stores persistent object to the memory.

Today some standard available in market for the object oriented database and most of today available object oriented database follows that standard [44][34][23][48]. The standards are discuss below,

2.3.4.3 Object Oriented Database Standards:

If you consider the object oriented database standard following two groups dominates for the standards of the object oriented database.

Object management group [43]:

This group proposed some standards for the Architecture and Tool to develop object-oriented system. It has also standard for the distributed object management. This is known for unified modeling languages.

Object Data management group [4]:

This group not developing standards for the object oriented database standards like object management group but it promotes portability and inner operations for them. ODMG provides standard for the object model, Object defined language and object and object query language. This all vary from the language to language.

Object oriented standards are proposed by ODMG and latest standard for object oriented database is ODMG 4.0 which is also called as "Next Generation Object Standards" [42]. Object oriented database is different from relational database and it is proposed in a way that it can adopt to verity of application requirements and it can be adopted by engineering field, multimedia field, statistic field, CIMS field, AI field, computer aided design / manufacture field, ERP fields, geographic information system field, integrated and embedded system fields and so on [36].

2.4 Agile programming

Agile programming is an approach to the project management in software engineering [40]. In 1970 Dr. Winston Royce proposed paper titled "Managing the Development of Large Software Systems" which is the root of agile programming language [40]. In this paper Dr. Winston introduced his thought on the waterfall. Basically building software is quite similar to the assembling the automobile [40]. In software each module should be isolated from another one it is same as automobile that each part is isolated from another part. But in other way one modules are dependent on another module and without proper combination of each module software does not work and this is also same with automobile assembling because even all part are isolated from each other though they are related to each other and proper combination is required [40]. In short agile programming is useful in project management and it includes schema design for programming and modularization of the software. This field (agile programming) also includes the refactoring of software. The refactoring in software engineering is described below.

Following topic shows what is refactoring in software engineering and what are the reasons (why? and when?) for the refactoring.

2.4.1 Refactoring in Software Engineering:

Refactoring in software terminology stands for changing the internal behavior of the software system without affecting the external behavior of the system [38]. Refactoring is the good way to clean up the existing code. Sometimes it also refers to improve the design of the system and software functionality [38].

Following are the main reasons that indicate why refactoring is required for any software [38].

- To improve the design of the software refactoring is needed. Sometimes poor design of the software may leads some problems in the software, in this condition need to change the design of the software [38].
- To make code easy to understand. Sometime code of software becomes so complex after longtime of the work in software at this situation it requires refactoring the software to make code easy for understanding [38].

- To help finding bugs. Sometimes after long development of the software. Software becomes so complex that, finding bugs becomes too complicated, at that time software needs refactoring to make it less complex for finding bugs [38].
- To make software run faster. Sometimes after a long time of development code becomes too complex and there are some unnecessary loop calls and function calls, so by refactoring the code this kind of false call can be removed and code becomes faster [38].

Following are the main reasons that indicates, when refactoring needed for any software [38]

- To add new functionality. Sometimes current design or existing framework of the software does not support some functionality, to add those functionality developers need to change design of software or need to do some changes in the existing framework which is refactoring.
- When need to fix bugs. This is another reason which requires refactoring. In some case to solve major bugs need to do refactoring in the software.

In this project titled "Refactoring Object Oriented Database", refactoring of the object oriented database is required to support some more functionality, which can cope up with schema changes in the class (Explained in section 2.7).

By considering above points of requirements of refactoring (when and why), it is required in "object oriented database" (software) to improve their design (first point of why) and to add new functionality for existing object oriented database (first point of when). For refactoring the object oriented database, I have changed the way it works. In short I have changed the design of the current object oriented database to achieve new functionality (Refactoring/schema evolution tool). My new architecture of object oriented database is explained in my design part of this report.

Now this report shows some features of object relational mapping and how it works.

2.5 Object Relational mapping:

Relational database has some advantage over other types of databases and today they are used widely. Same way object oriented programming language like Java, C++, C# has some advantages over other programming language so they are used very widely for programming. So in industry object oriented programming language is used for programming and to manage data they use relational database management system. In Object oriented programming it relates object to relational database Impedance mismatched [31]. According to IBM researchers in e-commerce application 30% of the code is used for database interaction. So by considering this kind of issue object relational mapping came into picture. Object relational mapping is Application Programmers Interface (API) that provides mapping of object oriented programming language to relational database. Today many object relational mapping APIs are available, one of them is hibernate framework.

2.5.1 How Object Relational mapping works?

Following figure shows how object relational mapping framework works? As in figure any field value of the class object can be stored inside the relational database table according to the given mapping files. Mapping only required once [31] and after that all insertion and selection of the objects, to and from the relational database done according to them.

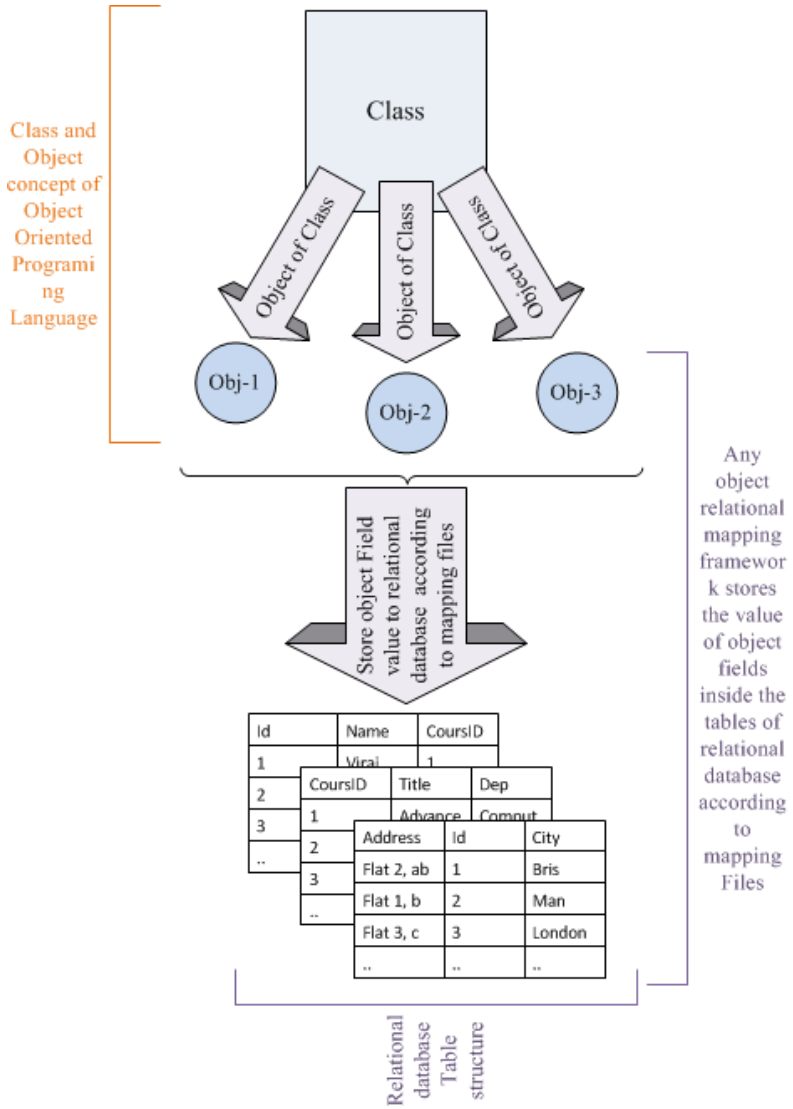


Figure 4 [Basic Structure of Object Relational mapping]

2.5.2 Features of currently available object Relational Mapping:

The Hibernate is the most used object relational mapping framework. This framework is available for the Java and .Net. Hibernate-framework works same as object relational mapping described above. Following are some features of hibernate.

- Hibernate ORM framework allows to create persistent class which follows object oriented features like inheritance, polymorphism, and composition [31].
- Hibernate does not require base class for persistency, further more Hibernate.org claim that Hibernate can do faster build procedure because it does not require any build time source or byte code generations. [31].
- Hibernate supports many features like lazy initializing, optimistic locking and different fetching strategies. [31]
- Hibernate is stable, reliable and high performance framework.

- Hibernate is highly extensible and very customizable [31].

Object Relational mapping are very handy while creating application in object oriented programming language with relational database.

Object relational mapping is one kind of object oriented database build on top of the relational database [46] while object oriented database is focusing on the high performance requirement. My approach of object oriented database is somewhat similar like object relational mapping (see My Design). [31] [46].

This project is focusing on, providing good schema evolution (refactoring) functionality for the object oriented database, which can change the stored object (inside object oriented database) structure according to schema changes in the respective class. So it is essential to understand the schema evolution inside the object oriented database and the class. Following two sections (2.6, 2.7) concentrate on that,

2.6 Schema evolution in object oriented database

After 1980, when "object oriented database" was proposed, a numbers of schema evolution methods have been constructed but most of them are unable to provide complete formal description of schema evolution [36]. In 2010 Jie Lin, Jinkun Yu, Zhiyong Zang presented paper [36] in which they describe the predicate formula for object oriented database schema evolution. This paper presents class evolution invariants and rules. Following are class-evolution invariants presented in [36].

- Class hierarchy invariants: Data model represented class hierarchy in object oriented database [36]. All class must have to maintain its structural characteristics of class hierarchy when changing the class evolution. This means if we change in hierarchy of class it will affect on the data hierarchy in OODBMS so try to avoid this [36].

- Name invariants: In object oriented database every class has unique name. So every class name must be maintained same.

- Origin invariants: In class, attributes and methods may have different origins. This point suggests that evolution in class must maintain the constant origins of attribute as well as methods [36].

- Full inheritance invariants: subclass should inherit all methods and attributes of the super class but they should not inherit attributes and methods of super class which creates name invariants and origin invariants.

Object oriented database provides schema evolution functions which changes the objects according to the schema changes inside the respective class [20][23][34][44]. This functionality includes renaming class, adding field, removing field, and renaming fields.

2.7 Schema changes (Schema evolution) for class in programming language

Schema changes in class usually occur during the refactoring of the project. These schema changes includes the renaming of the class, removing fields from the class, adding fields to the class, moving fields from one class to other class (especially in inheritance case) and copying fields. It also includes some class hierarchy changes like removing class from the hierarchy,

adding class to hierarchy and changing hierarchy sequence. By knowing the schema changes inside the class it is easy to understand how it can affect to the object oriented database.

3. State of the Arts in object oriented database

As this project is all about object oriented database, this part of my report shows, what kind of work is already done in the area of "object oriented database". The above section of this report explains what is object oriented database and how it works? This section discuss about some of the most used and reliable object oriented database which are available in market.

Today many object oriented database are available in market for different object oriented languages. Some of them are versant object database, db4o, Objectivity/DB, Object Store etc. Today most popular and open source object oriented database for java programming language is db4o [20]:

3.1 DB4Objects (db4o)

DB4O is the open object oriented database which is available for the C#, java. Currently this database is also available for the android operating system in the mobile phones. DB4O database is available for the client and server applications and it also supports the transaction process. To Store and retrieve the objects db4o supports query by example, native queries and SODA queries [47].

Following figure shows the architecture of the db4o object oriented database. As you can see it is quite complex structure for object oriented database. This architecture shows how the persistent object in ram can be stored in file system and before storing objects in to file system db4o conforms ACID properties of the transaction.

Reference system is used for the indexing and querying on the objects which are already stored. Class metadata is the structure which checks for the schema evolution in the class that are associated with the object and according to that it change the class data which are stored in to database.

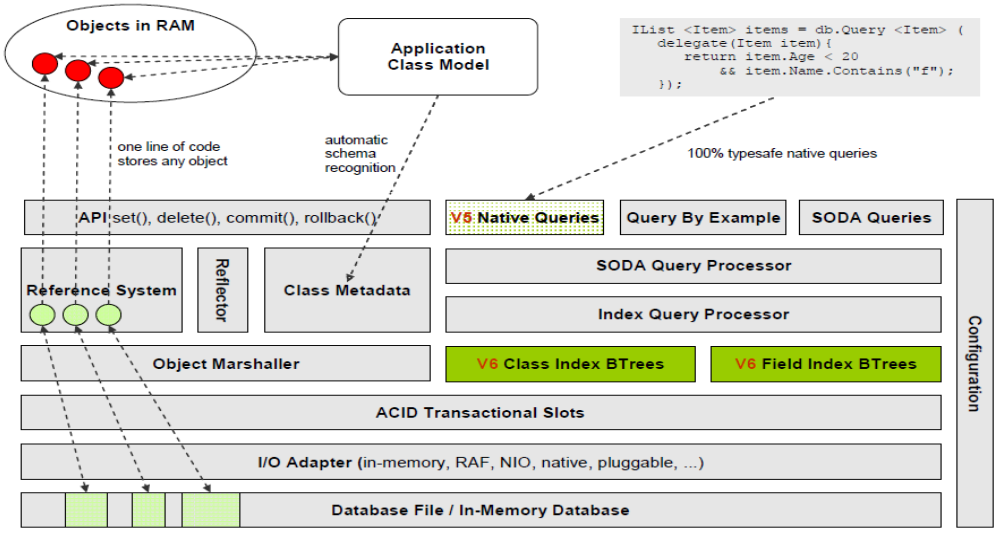


Figure 5 [Architecture of db4o (copied from [22])]

Another most popular object oriented database is versant object database. Following are some discussion about versant object database architecture and some feature of them.

3.2 Versant object database

This object oriented database is available for the different object oriented languages like Java, C++, .Net (c#). It has some key features such as, it supports the standard, seamless database distribution, dynamic schema evolution, low to zero administration needed, it has end to end object architecture, support for good concurrency control, and it supports multi threading and multi session [44].

It has some key benefits like, it stores object faster, it has 10X performance of RDBMS, it cuts development time up to 40% because of direct storage of the object and it also leads towards the lower server hardware costs [44].

Following figure shows the versant object oriented database architecture.

As shown in figure it takes object from client/server's cache and store it to the database. It also provides some functionality like administration control, monitoring consol, object inspector, and XML toolkit.

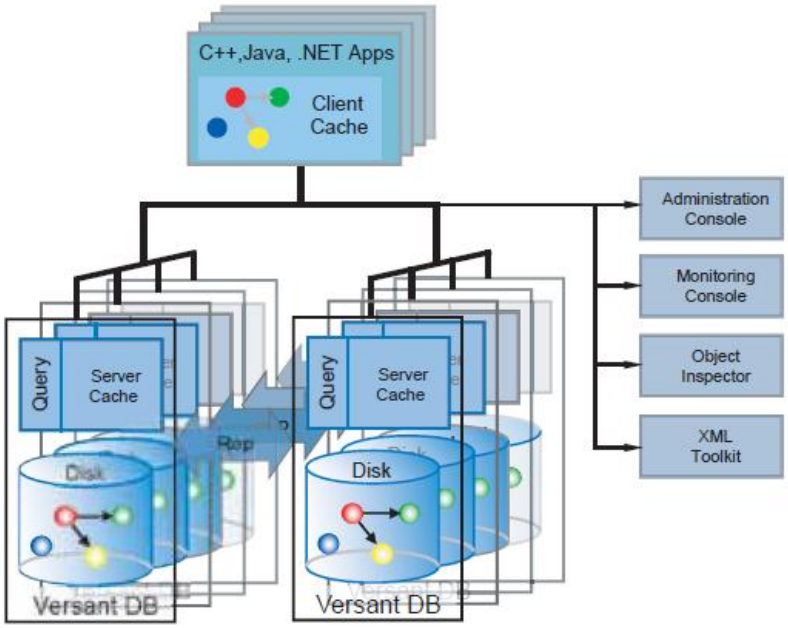


Figure 6 [Architecture of versant object database (copied from the [44])]

3.3 Brief about other object oriented database

Apart from above discuss object oriented database, there are some other object oriented database that are also available among them ObjectStore and Objectivity. Both of them have almost same functionality like above discussed object oriented database [34] [45].

Following are some advantages and disadvantages of currently available object oriented databases.

3.4 Advantages of object oriented database [20]:

- No mapping and conversion needed: This means that no need to relate object oriented languages to the relational databases. If you store object oriented data to the relational database some kind of relation needed, which takes values from the object's field and add it to the relational database's tables. While object oriented database directly stores object (instance of class) to the database therefore, there is no requirement of doing any mapping.

- Normal behavior of class is needed to make object persistent, which means we don't have to change class for its object's persistent behavior. To make object persistent no special requirement is needed inside the related class.

- To store an object of the complex structure class, just one line code is required for writing: As Object oriented database stores object directly to database, therefore, no need to worry about the object complexity. The situation of complexity may occur due to the inheritance of the class. As object oriented programming languages provide method to store object to the database, so user only need to call that method for any complex object (to store it in database).

- This database not only works in local but it also runs in client server environment. Today available object oriented databases provides client/server architecture support. So object oriented database also works with client server applications.

- This database also supports transaction model based on ACID property. Most of the available object oriented database provides transaction facilities base on the ACID property (As explained above inside relational database).

- This database supports Database versioning and automatic management for its schema of database: This means that most of available object oriented database can manages versioning between old and new version of class, whose object stored inside the database. Some of available object oriented database can deal with automated object schema change according to change in their class, but this functionality is limited for some type of schema changes.

- This database is integrated with native garbage collector: Today available object oriented database come along with functionality, which collects old and unused objects stored inside them and destroy them.

Above advantages makes object oriented database more convenient than the relational database for object oriented programming languages due to some reasons which restricts the use of the object oriented database. Some of them are discuss as follow.

3.5 Loose points of object oriented database [6] [47].

- Less standardize: Even though there are some standards for the object oriented database as shown above (section 2.3.4.3) but if you compare object oriented database with relational database, it is less standardize then RDBMS.
- Less Tool Support: Due to less use of the object oriented database fewer tools are available for them.

- Major vendors have still not except the object oriented database completely and less marketing is one reason of less usage of object oriented database.
- Today object oriented database can deal with the schema change in class and change their object schema accordingly in database and most vendors provide this functionality which works automatic, this functionality called as the scheme evolution functionality. But this schema evolution or refactoring functionality is limited in today available object oriented database and this is one reason of the less used of object oriented database.

4. Problem statement:

Among the above discuss reasons which restrict the usage of object oriented database, my project is concentrated on the removal of the last loose point, which is limited functionality for schema evolution or refactoring in object oriented database.

Main reason behind the less functionality of refactoring in object oriented database is persistent storage of object inside the permanent memory [22] [32] [44]. Because of the direct storage of the objects in to memory, it is hard to change structure of the object according to class structure change and this is the reason why all object oriented database can not deal with schema changes easily. To remove this drawback, somehow I have to provide partial isolation of stored objects from their class and by doing this I can manage class and confirm that object are not dependent on each other and I can provide tools in between them, which can deal with schema changes in object according to class.

To provide partial isolation to the object stored in database from the class associated with it, I have proposed a new architecture which can store value of fields of the object, instead of the object storage and at the time of reading object, it converts all stored field values to the object and returns it. I am also storing the class structure and the class records which help me to deal with the schema changes inside the class. This approach gives partial isolation to the stored object from its class structure and at the time of schema change in class my provided tool can deal with stored object's field's value.

If object oriented database come up with the good refactoring (schema evolution) functionality, then object oriented database can replace relational database in big projects.

5. My Design

In explanation “my program” indicates my demo implementation for my approach.

“My demo” word in the following description indicates my implementation for this approach.

‘oo’ or ‘OO’ indicates ‘object oriented’ in following discussion and ‘OODB’ indicates “object oriented database system”

5. 1Explanation of my design:

There are some schema evolution issues with Object persistent storage (OODB) and this proposed design is to deal with class schema change easily. I have implemented demo to show how this approach can work. This approach is implemented using java language and my demo implementation only works for java language. My demo will work with basic functionality of the querying and Refactoring.

5.1.1 Why Java?

As my project is object oriented database it can only work with object oriented programming languages [18]. I have some options like Java, C++ or C#. But I have selected Java due to its benefits such as: ‘Simple’, ‘Secure’, ‘Portable’, ‘Object Oriented’, ‘Robust’, ‘Multi-thread support’, ‘High performance’, ‘Distributed’ and ‘Dynamic’ properties. [11]. Among all properties of java object oriented, portable and dynamic are the main properties that encouraged me to choose Java for this project. Another reason to choose java is, it is open source technology and currently it is used very widely compare to other object oriented languages.

5.1.2 Basic Structure of my proposal:

To make refactoring class (cope up with schema changes in class) easy this project design is proposed by taking inspiration from the object relational mapping [31]. My proposal is for object oriented database which can provide all the function that are currently available in all object oriented databases [20] [22] and also provides some additional effective refactoring (schema evolution) functionality to remove major drawback of currently available object oriented database. This design proposes inbuilt file system to store value of object instead of persistent storage of the object.

To store value of the object’s fields I am proposing three files. One of them is used to store all class records, second one stores structure of the class and third one contains the values of different objects fields. Following are three files which I have estimated.

1. Class Record File
2. Class Structure Record File
3. Object Value File

Class Record File:

This file contains the record of all classes which are stored in database. In this file only one entry is there for each class with its unique class number. This class number is referred by other two files (Class Structure Record File, Object Value File). When user stores object of class into the database my program will check for previous entry of associated class file with that object. If there was no previous entry for associated class then my program will enter class entry

with unique class number, otherwise it will take the unique class number to store value of the object. “Figure 7” shows the structure of file.

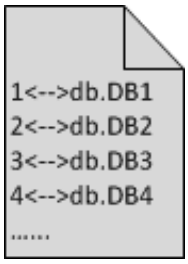


Figure 7 [Representation of Class record File]

In “Figure 7” the number indicates the unique number for the classes and name indicates the class name with packages and “<<-->>” symbol indicates the separator.

General structure of the record in this file:

[Class Unique Number] + [Separator] + [Class Name]

[Class Unique Number] – represents string which contains unique class number

[Separator] – represents the separator to separate class unique number and class name, in this file it is “<<-->>”

[Class Name] – represents string which contain class name.

Class Structure Record File:

This file contains the structural record of the classes. This file contains only one entry for each class with its unique class number which is in Class Record File (previous one). This file can be useful to find class schema changes. ‘Figure 8’ shows the structure of the file.

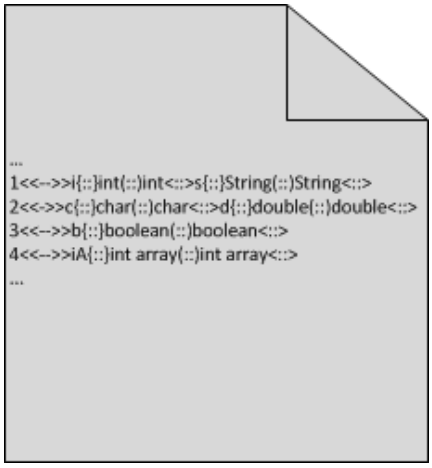


Figure 8 [Representation of Class Structure Record File]

In ‘Figure 8’ the number indicates the unique class Number which is declared in Class Record File. In File “<<-->>” is the separator to separate class number from structure, “<::>” is

the field separator to separate different fields and "{::}", "(::)" are the field value separators used to separate Field name, Field type and Field Generic type

General Structure of record in this file:

[Class Unique Number] + [Separator] + { [*Field Values*] + [Fields Separator] + [*Field Values*] + [Fields Separator] +… } where [*Field Values*] = [Field Name] + [Field Value Separator1] + [Field Type] + [Field Value Separator 2] + [Field Generic Type]

[Class Unique Number] – This represents the unique number of the class

[Separator] – This represents the "<<-->>" string as separator.

[*Field Values*] – This is a combination of field name, field type and field generic type of class.

[Fields Separator] – This represents "<::>" string as field separator.

[Field Name] – This represents the field name of the class.

[Field Value Separator1] – This indicates the "{::}" as separator.

[Field Type] – This represents the field type of the class.

[Field Value Separator 2] – This indicates the "(::)" as separator.

[Field Generic Type] – This represents the generic type of the class field in the class.

Object Value File:

This file contains data of different objects for different classes. This file contains multiple entries for one class objects. This file uses to get track of the object record. This is main database file which is useful to store object's field's values and get back those values to the objects. Each entry has one unique number which is object unique number. 'Figure 9' shows the structure of the object value file with dummy data.

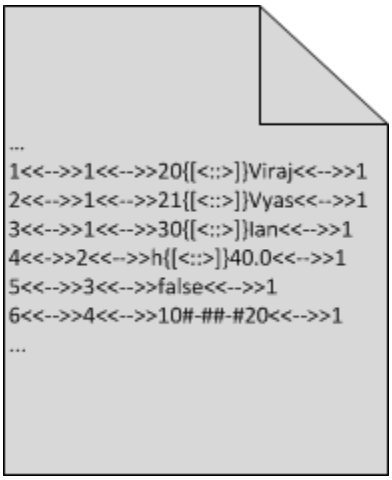


Figure 9 [Representation of Object Value file]

In ‘Figure 9’, the first number shows the object unique number, “<<-->>” shows the separator. Second number in ‘Figure 9’, is the unique class number which is linked with class record file. “{[<::>]}” shows the field values separator.

In this file values are in synchronized with class structure file, for the field sequence i.e. First value in this file belongs to the first [*Field Values*] inside the class structure file. Last number shows status of the record.

General Structure of record in this file:

[Object Unique Number] + [Separator] + [Class Unique Number] + [Separator] + [*Values*] + [Separator] + [Status] where [*Values*] = {[Field1 Value] + [Field Value Separator] + [Field2 Value] + [Field Value Separator] …}

[Object Unique Number] – This represents the unique number for that object value.

[Separator] – This indicates “<<-->>” string which is useful for separator.

[Class Unique Number] – This represents the unique number of the class which is mentioned in the class record file. This points the class whose object is stored.

[*Values*] – This field contains all filed values of the object concatenated with the separator. This values sequence is synchronized with the sequence of [*Field Values*] of the class structure file.

[Status] – This represents the status of the record. Currently I am not using this field in my demo but it can be useful in future development of this approach.

[Field1 Value] – This string represents the field’s values.

[Field Value Separator] – This indicates “{[<::>]}” string which is useful to separate different field’s value.

So in my proposal I am going to store values of the object’s fields inside the above discussed file system, instead of storing object directly to database as persistent storage. The above mentioned three files are useful to store fields’ values of the object. At the time of fetching (selecting) object from database I am creating an object of mentioned class, according to stored values in the files. Both Insertion and Selection of object approach is discussed below in detail.

5.1.2.1 Basic functionality with my system

Following figures and their descriptions show how basic functionality can work using above projected design of the file system.

Insertion of Object:

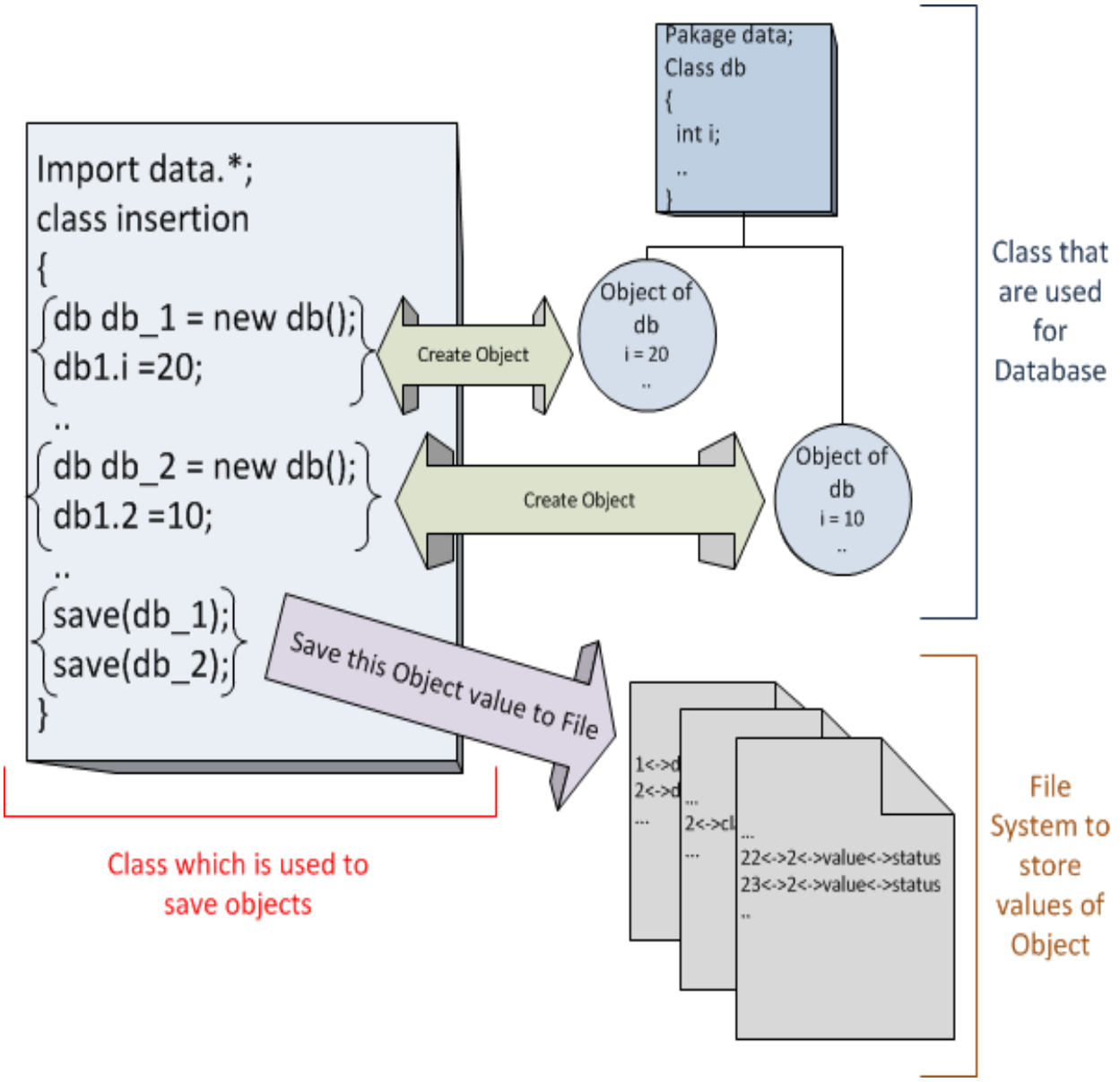


Figure 10 [How Insert Functionality can work with my approach]

'Figure 10' shows how basic insertion can work using my proposal of Object Oriented database.

As shown in 'Figure 10', one class named 'db' whose objects are used to store in database. Another class named 'insertion' is useful to create the object of 'db' class and store it to the object to database using the method of my demo API. As you can see, 'Insertion' class creates two objects of the db class and assign different value to its field named 'i'. Then it calls method named 'save' to save that object to the database.

This 'save' function takes object as the parameter and by performing logical operations on the object it stores class' field's value to the above discuss file system. By doing this my approach becomes different from available object oriented database [20] [22] [32] because it didn't store object direct to database system. This approach is useful to make schema evolution functionality more flexible.

Retrieving object from files:

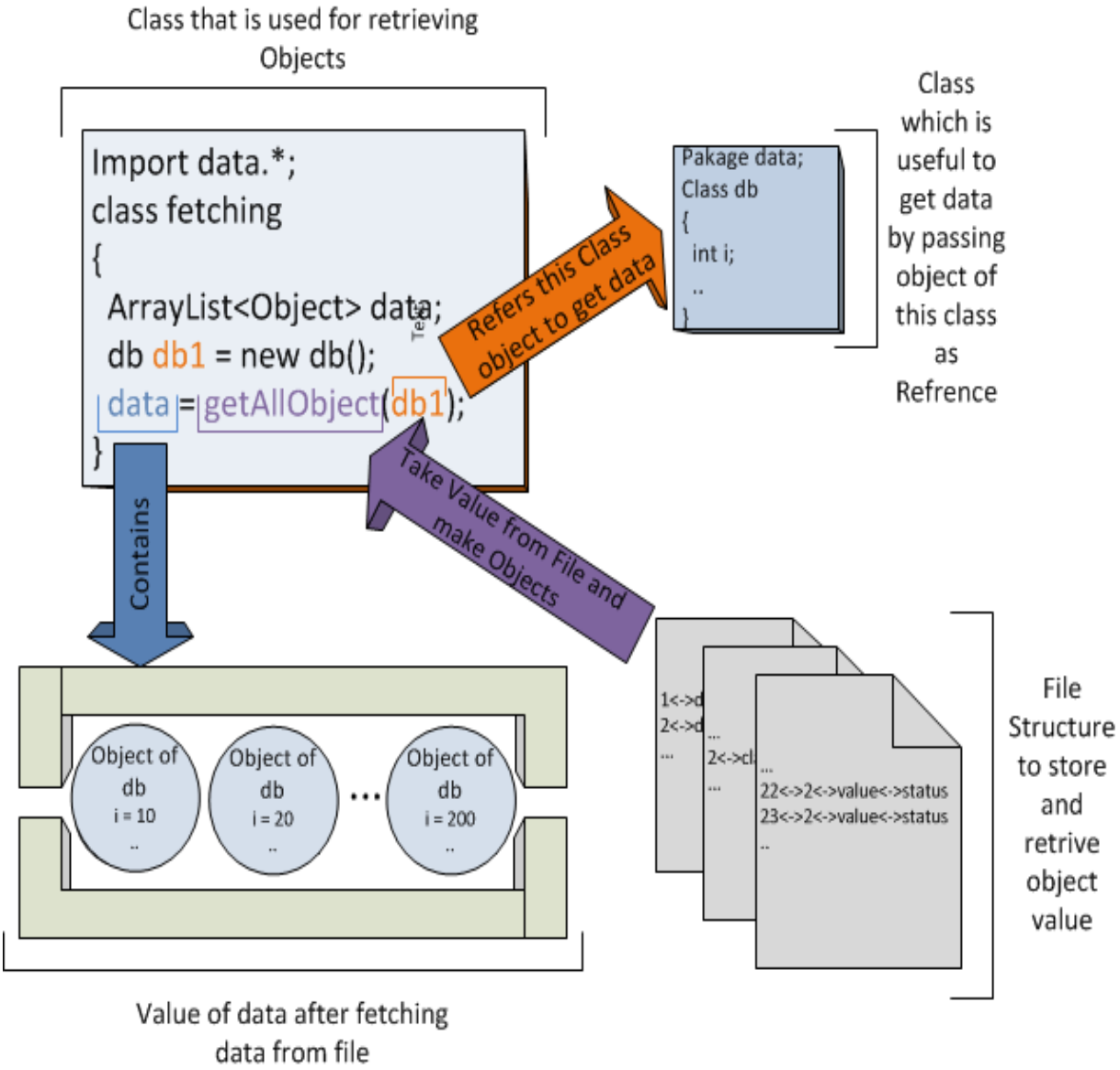


Figure 11 [Retrieval of Object from File]

'Figure 11' shows how object reading can work in my proposed design of Object Oriented database.

As shown in 'Figure 11', 'fetching' class calls method named 'getAllObject' and pass object of db class as parameter. This passed object (as argument) indicates that the user wants to read all objects of 'db' class which were stored in database. To do so my approach is to read field's values from the file system (as explained above) and using Java reflection create objects of 'db' class and assign values to it [12]. By using this technique user can read data as object instead of simple value fields and thus my approach work toward the object oriented database [18].

5.1.3 Tools

To make database more useful and convenient I am proposing two tools with this approach. Both these tool makes my new approach towards object oriented database more user friendly. In my demo I have provided these tools as API but GUI representation of this tool make this approach towards object oriented database more attractive. Tools are as follow,

1. Query tool
2. Refactoring tool

5.1.3.1 Query tool

This tool mainly deals with insertion of data as object(s), to update object(s), to delete object(s) and to read values as object(s). Following four functions is also provided in my demo implementation of this approach.

Insertion Object(s):

This function inserts field values of object to file system according to the proposal. I have provided this function in my demo API which takes object as argument of function and stores it to files. Insertion function takes object as the argument and stores the field value of the object using java reflection API [12] .

Update Object(s):

This function can update already existing value(s) in the file system. I have provided this function in my demo API which takes parameters as follows class object, field's name, field's new value, condition field's name, condition field's value. This method checks the given condition fields against condition fields values and identifies which object needs to be updated and updates the fields value as per given name in second parameter.

This functionality can be provided in different form as well. i.e. same as above function but without using condition field, to change all field value according to class object.

Delete Object(s):

This functionality can delete records from the file system (as proposed above) according to the passed condition and the object of the class as a parameter of the function.

I have presented this functionality in two ways. One of them only uses object to refer to associated class and delete all records associated with that class inside the database (file system in our case). While the other way, removes the record of the class from the database (File system) according to the condition fields and its values passed to the function.

Retrieve Object(s):

This functionality is useful to read object from the database (file system in this case). In my demo implementation I have provided some functions which read field's values from the file and create the object using those values.

This functionality can be presented in two ways; one of them only uses the object of class as reference and retrieves all objects that are associated with object's class. These are passed as

arguments. Second method uses condition fields and their values to filter data. This functionality can read data (field's values) from the file system (database) according to conditions that are passed as arguments of the function, and assign those data (field's values) to the dynamically created objects of the class whose object was passed as argument of retrieval function.

5.1.3.2 Refactoring tool

This is the main and unique feature of my project. The main aim of this project is refactoring already existing object oriented database architecture and propose new architecture for object oriented database, so we can achieve a more easy and robust refactoring tool. This tool is useful to deal with object oriented database changes, according to class schema changes inside the class i.e. this tool changes the name of the class in an object oriented database file system if the class name is changed in the program. I have provided this tool as API in my demo implementation but GUI representation of this tool will be handed to users. In my demo implementation I have provided some refactoring functionality which includes rename class, rename class fields, add fields, remove fields, move fields, copy fields and remove class from hierarchy. All are explained below.

Rename class:

By using this functionality, user can change class name in object oriented database (file system in this case), according to the name change in class. After renaming the class, user will receive the record which was stored using the previous name of the class. To implement this functionality is hard in the object oriented database which uses persistent storage of the object, but in this proposed design, it is easy to change the class name [18]. To implement this functionality in my proposed design only thing needed to do was to replace the old name with the new name of class inside the class record file (one of my proposed file).

This rename class method in my demo takes two arguments as parameters. The first one is the old class name and second one is new class name. After calling this method, user will get back all record using new the class name. To get back objects of renamed classes, which were stored using the previous name of that class, user only needs to call one method from the API (demo) named renameClass and after that user will get all the record from next selection function using new class name.

Rename class Field:

A feature of this functionality is that it allows the user to change filed name, to the OO database (file system in this case) for specific class after changing its (fields) name to the class within the programming language. In my demo I have done this in Java).

To accomplish this functionality I have provided a function called "renameClassField" in my demo implementation, which takes parameters like 'class name', class field 'old name(s)' as an array and class field 'new name(s)' as an array. This function changes the class' field's old names to new names in the database files. By using my design this functionality is easier to implement. The only thing that is needs to be changed is the field name in object structure file (one of my proposed files system).

Add Fields:

This functionality helps user to add field with specific value/default value to the object oriented database, after adding new field to the class, in programming language.

In my demo implementation of the object oriented programming language, I have provided the functionality to add field to the object oriented database (file system in this case) for all the objects (object field's values in this case) that are stored according to previous structure (stored in the object structure file in this case) with default value or with user defined values. This function takes the class name, field(s) name and field(s) value as argument and do the following steps to add field for class in my approached object oriented database.

NOTE: The argument for Field(s) value is optional. If it is not passed, then field will be assigned a default value). To implement this functionality in my design, I needed to change two files. They are class structure file and object value file. In class structure file, needs to add new entry of [*Field Values*] according to new field for given class (in argument of the function). In the object values file, I needed to append new values for the new field to the all record which are associated with the class (given as parameter of function).

Remove Fields:

This functionality helps user to remove fields from the object oriented database, after removing this field from the class in the object oriented language.

I have provided method in my demo, which removes the field from the oo database according to given parameter to the function. This function takes class name and its field name as the parameter. According to the given parameter, it removes the field from the associated class structure in the class structure file and associated field's value of objects from the object value file.

Move fields:

This is a unique feature provided by my demo API for my structure. The user can move fields from super classes to sub classes with values by using this functionality.

After doing change in class schema (moving field from one class to other class) user will apply this function for database and this function do same changes in database (in this case need to change value in above proposed file system).

Currently this functionality only works when the original class and destination class have the same record of objects in the object oriented database. In this case, this function only works when both classes have same number of records in the object value file. If both objects of classes don't have same number of value in database then there might be chance of data lost.

This functionality is provided by the function 'moveFields' which takes origin class name, new class name and array of field(s) as parameter and cut fields from origin class and paste that field to the new class. This functionality is possible using my proposed design (store values in file system instead of persistent storage) of object oriented database.

This method changes two files to move field(s) from one class to other. In class structure file structure changes according to passed argument of the function and in object values file valued of object field changes according to new structure of classes in class structure file.

Copy fields:

This functionality is quite similar to above functionality. Only change in this functionality is that this function doesn't delete fields from original class records so this function keeps original copy as it is just copy that field to new class's (passed as argument) object value.

This function takes parameter like old (origin) class name, new class name, and field(s) name. This argument indicates that user wants to copy passed fields from old class to the new class with values if possible.

This function changes two files. One of them is class structure file and other one is object value file. In class structure file new class structure need changes to add new fields. This structure need to add new [*Field Values*] for new field. In object value file need to change new class' objects' fields structure according to structure change in the class structure file.

Remove all super classes:

This functionality removes the reference of the super classes. In my demo API this function is named as remveAllSuperClasses which takes class name as an argument and removes references of all class inside the database.

This function is useful to make class independent from the hierarchy.

Above mention functions are implemented in my demo implementation of my approach for the object oriented database. But in the future work/proposal I have discuss some more functions for the refactoring tool. After implementation of all these proposed refactoring tools object oriented database will become the most handy oo database, which can deal with the schema changes in the class [20] [22] [23] [32]. All refactoring tool functionality becomes possible because of doing partial isolation of the object from the class. This partial isolation achieved by storing object's fields' values to file system instead of the persistent storage of the object to memory.

5.2 Challenges and its solution during demo Implementation

There were some challenges that I have faced during the implementing demo for my proposal. By researching on Internet and taking help of my supervisor, Mr. Holyer I was able to overcome that challenges. Some of them are mentioned below.

Design proposal:

My first challenge was how to do refactoring of object oriented database? Refactoring in Software Engineering means restructuring existing body or code to change their internal behavior without changing their external behavior [33]. So to do refactoring object oriented database somehow I have to change internal behavior currently available in object oriented database. As class schema change doesn't make object of that class stored in oo database outdated, this is the main objective behind refactoring. I have to propose a new approach for the object oriented database so that class and their object in oo database, both are isolated from each

other so we can do schema changes in the database according to the schema change in related class.

Now my next challenge was: what can be the new proposal for the object oriented database so I can get partial isolation for the class and their objects inside the oo database. I did plenty of online research for this and lots of discussion with supervisor and finally I came up with the above proposal for the object oriented database. In my proposal, objects and classes both are isolated from each other and I can do the same changes to my proposed file system (for object oriented database) as the schema changes in class.

Read object:

When I have decided to go with approach that stores value to the file instead of persistent storage of object, very first challenge I got is, how can I read object values. I researched about it and I came to know that Java programming language is providing Reflection feature, which is useful to read object field values.[12] I have implemented a demo of that feature and after getting success in the demo I have implemented an object oriented database version of the demo.

Create object dynamic and assign values to it:

Next challenge that I have faced is, how can I create object of class using its name? And after creating that how can I assign value to it?

To achieve these challenges I have done some research on Internet and I came to conclusion that using "forName" method of "Class" class in Java I can load class dynamically, [12] [13] and after that "newInstance" method of class can allow me to create new object of that dynamically loaded class [12] [13] . To set value of the fields for dynamically loaded object I can use reflection functionality of the Java [12].

Implement Inheritance functionality:

After successfully completing above challenges, one major challenge was to implement inheritance functionality in my demo implementation of the object oriented database. I have to implement solution for this challenge for two scenarios.

First one is while user saves the object. And second when the user is trying to retrieve that value.

Save object to database:

While user tries to save object inside my demo object oriented database, according to my proposal, I am saving the value of that object fields instead of the persistent storage of the database. This was actually a challenge, to implement inheritance functionality in demo because in persistent storage this implementation done automatically (by oo programming language). [11] However, in my demo I have to implement some technique to deal with this. After conducting intensive research on this challenge and discussing with my supervisor finally I came to the solution, which can deal with this challenge.

My solution is to add extra string at the end of the [*Values*] string in object value file of my proposed file system. This extra string is combination of the static string, named

”supreObjectValueNumber:”, and super object value unique number. This super object value unique number refers to the fields’ value of super class object.

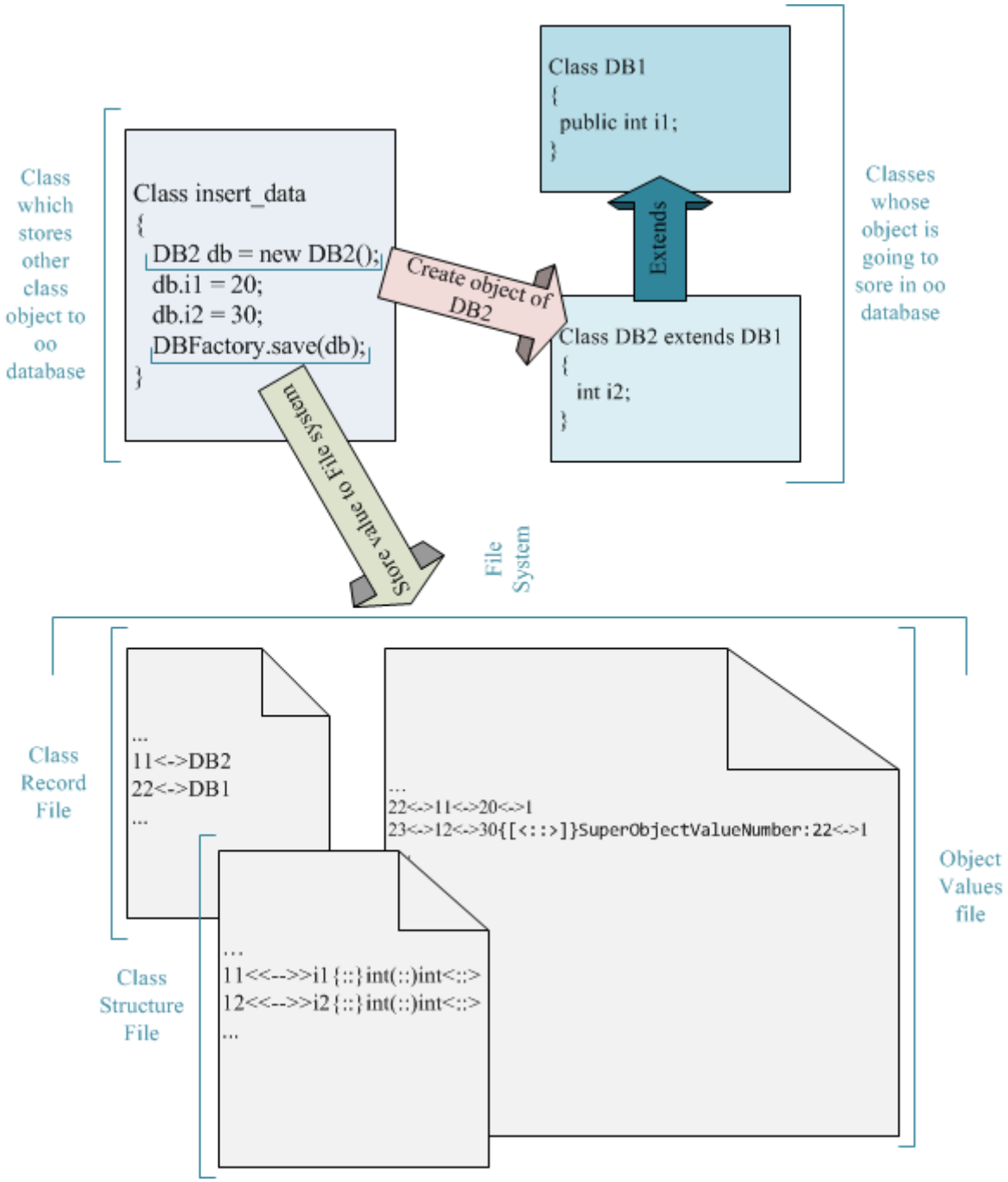


Figure 12 [How inheritance can work with my approach]

Figure 12, shows my approach on dealing with the inheritance class. In Figure 12, class ‘DB2’ extends ‘DB1’ class and ‘insert_data’ class inserts the object of ‘DB2’ class in my proposed object oriented database. Three file shows that how my system handles the inheritance.

As shown in figure 12, while 'insert_data' class calls the function to insert data into the oo database (in this case file system) and as you can see in figure 12 how data enters into the files system. My demo API stores both class records to the 'class record' file so you can see the entry of both class in the class record files in above figure. My demo program also enters structure of both classes (DB1 and DB2) into the class structure file according to above explained format. As you can see the entry of classes' structure in to class structure files in figure 12. Figure 12, shows how data of the classes' fields enters in to the object value file by my demo API. My demo implementation of the Object oriented database first enters the value of the super class to the database so as you can see the first entry for the DB1 class in figure 12. After insertion of parent class' object's field's value to the object value file, it inserts the child class object's field value and at the end gives the reference to the parent class's object record using string "superObjectValueNumber:" and after that it concatenate unique number of object value for parent class. In the figure you can see the following string in the object value file:

23<<-->>12<<-->>30{[<::>]}SuperObjectValueNumber:22<<-->>1

This is the record for the object of the DB2 class's fields. Where,

"23" indicate the unique number of that object value.

"12" indicates the unique number of class. In our case 12 refers to the DB2 class (from class Records file).

"30" indicate the value of the i2 field of the DB2 class. This value comes from the class structure file.

"SuperObjectValueNumber:22" indicates that this record is associated with the other record of the object. 22 indicate object's record unique number. In this case it refers to the above record which is the record of the DB1 class.

"<<-->>" is the separator to differentiate different values like object record unique number, class unique number, fields' record, and status.

"1" indicates status number.

Read object from database:

Another challenge that I have got to implement inheritance is, how can I fill value to super class fields using subclass objects? And for that I have done internet research and after that I come to know that it is possible to do that but somehow you need to get fields of super class. Following are my technique to deal with this problem.

Now using above shown structure for inheritance (figure 12), at the time of the object reading functionality, my demo API somehow manages to read value of the parent class's field for inheritance structure of the class.

To read the parent class value my demo implementation uses the string "SuperObjectValueNumber:[unique number of object record]" and using this [unique number of the object] record it can read the super class field's values for given sub(child) class.

One more challenge I faced was, how can I get field of super class using object of sub class? But after some research I come to know that java itself provides some functionality that is useful to get fields of super class [12] [13].

Using this technique I can deal with inheritance feature of the object oriented methodology. This leads my database towards the object orientation.

To implement inheritance functionality was one of the major challenges that I faced during the development of the demo for my approach of the object oriented database.

How to get access to private and protected fields?

While I was trying to access private and protected fields of the object it was throwing Illegal access Exception at beginning. Then after some research I came to know that java provides a method setAccessible(boolen), which is useful to access private and protected fields of the object [12][13].

How to deal with array fields?

How to store array field values was the next challenge for me. For that I used separators to concatenate array values and stored those values as string. At the time of reading back I split that string and assign values to array back.

5.3 Current Limitation of the demo:

My demo implementation has some limitations. Main reason behind this limitation is lack of time. I have some future proposal to overcome these limitations. Currently my project has following limitations.

Only deal with the limited number of field types:

Currently my project only deals with limited number of field type. It deals with field types like String, int, double, boolean, char, float. To show that this approach can deal with array as well, I have implemented demo which deals with array of string and integer only but this approach can support all types of the fields for array [17]. In short my demo implementation currently deals with only primitive types and array of primitive types [28]. Currently my demo not dealing with the object type and the collection of the objects but all type can be supported using this approach and I have discussed some idea in the future proposal to explain how all field types can be supported in my proposed structure?

No support for enum type:

Currently my demo does not support enum type. But java provides some functionality in reflection API which can be useful to deal with the enum type in my proposed object oriented design [15] [16]. I have to manage file system so that it can deal with enum type and for that I have one proposal which is quite similar to the way that stores super class object field value (in inheritance functionality). Only thing I have to do is to append enum class number (link to Class record file) to the enum value (object value file) so that I can get track on appended field value coming from given enum class (by referring Number which is appended).

Limited with functionality:

Currently this demo is limited with the Functionality. I have provided only few functions that are mandatory. But some more handy functions will make this approach towards desirable Object oriented database. I have proposed some functionality that can be useful to make approached database more popular in future proposal.

No support of constraints:

Currently my demo implementation of my approach on object oriented database does not support the constraint functionality. In future proposal I have discuss some idea that can be useful to implement constraint functionality in my approach of object oriented database. I have discussed about three type of constraints i.e. unique constrain, primary key constraint, and foreign key constraint.

No automated schema change detection:

Currently demo implementation of my approach on object oriented database does not provide automated schema change correction functionality. This functionality can change structure of the class in class structure file (one of my proposed file) and related other files automatically if someone did minor change in class fields. Minor change includes possible field type change, add new field, remove field and class field sequence changed.

I am getting the difference between current class structure which may be changed due refactoring to the existing structure stored in class structure file. By doing some comparison I can get track of the fields that are changed or newly added. Then according to that I can do related refactoring in my file systems. In future proposal section I have discuss some idea to implement this automation.

Field sequence problem:

According to my proposal of the file system all files are related to other file. As I discuss earlier object value file is dependent on the class structure file because object value files stores the value of the fields in one string with some separator and sequence of the fields value is depend on the class structure file's [*Fields Values*]'s sequence.

Now if some user changes the sequence of the field inside the class. Somehow I need to track changes of the fields and need to apply it on the class structure file for the [*Fields Values*]'s sequence and same change need to apply in the object value files for the values. Otherwise new entered record in to object value file creates the problem because of the new sequence of the class fields.

There are some solution to deal with this situation and one of them is to create sequence of the object's fields' record according to the sequence stored in to the database so no need to change both file for this problem.

5.4 Proposal for improvement

Above I have presented a new approach for the object oriented database and I have also made demo implementation of that. My demo implementation has limited functionality, so I am

proposing new functionality that can be added to my demo implementation of the object oriented database. In following discussion I am going to talk about how functionalities can be added in the object oriented database developed in this project, features like, GUI representation of the tools, constraints, automated schema changes(Smart schema change), log.

GUI representation of tools:

To make my approach more users' friendly GUI tool may be handy. GUI can be provided for the both query tool and refactoring tool.

Query tool GUI representation:

GUI tool can be provided such that user can easily do the functions like, insert object, remove object, update object and select the objects. The tool can support query like sql, so migration from relational database to object oriented database can be made easier. Sql can be supported to maintain the database because of my different approach for the object-oriented database.

SQL support can be added in the demo implementation by adding new feature tht can call function according to SQL requirements, this will allow user to use SQL to maintain data (in this case objects).

Refactoring tool GUI representation:

To make class structure change in to database more easily this tool might be useful. Main idea behind this tool is to change class using this tool will change the associated database with that class. This tool is unique functionality for the object oriented database [20] [22] [23] [32].

This tool will provide all the facility like add fields, remove fields, move fields, copy fields and some function that are useful to change the hierarchy of the class. This tool is useful while user refactoring their code and change schema of the class. This tool will help to change class as well as the related oo database.

Add Different constrains:

To make my approach competitor to currently available object database, it should have Different constraints [20] [21] [23]. I am going to propose the three different constraints for my proposed design of the object oriented database and they are unique constraint, primary key constraint and foreign key constraints.

Following are some description of the constraints after that I have mention approach how it can be implemented using my design of the object oriented database. Before going for each constrains individually, firstly see, the general type of structure change needed to add constraint functionality in my proposed design of oo database.

General structural change: Following are the basic changes that are needed in the "class structure file" to add any constraint in my proposed object oriented database. Before going through changes, let's look at the current format of the "class structure file" in my proposed design of object oriented database.

Current General Structure of the class structure file:

[Class Unique Number] + [Separator] + { [*Field Values*] + [Fields Separator] + [*Field Values*] + [Fields Separator] +… }

Where value of [*Field Values*] is as following.

[*Field Values*] = [Field Name] + [Field Value Separator1] + [Field Type] + [Field Value Separator 2] + [Field Generic Type]

Now to add constraint functionality I am proposing to change format of the class structure file and they are, one new string (constraint information) can be append inside the [*Field Values*] after the [Field Generic Type] and that is, [Field Constraint] and it can be separated by the [Field Value Separator 3]. [Field constraint] is a string "constraint: [Constraint Name]". So after implementing this functionality structure of [*Field Values*] is as following,

[*Field Values*] = [Field Name] + [Field Value Separator1] + [Field Type] + [Field Value Separator 2] + [Field Generic Type] + [Field Value Separator 3] + [Constraint Name]

[Field Value Separator 3] can be any separator that can separate the value of the constraints to other value strings.

[Constraint Name] indicates the present of the constraints with that field and it might be optional field. If [Constraint Name] is available then it indicates which type of the constraint associated with the field which is represented by that [*Field Values*].

It might be possible that one field contains more than one constraint. In this case [Constraint Name] contains all constraint representation by separating them by comma or some other separator.

Above file structure change is a general change to add constraints in my proposed design of object oriented database, but to add different type of constraints need to implement new functionality to fulfill those constraints. Following change needed for the implementation of the individual constrains. Following are proposal for the unique constraint, primary key constraint and foreign key constraints.

Unique Constraint:

This constrain prevent user to add same value for different object's same fields in to object oriented database. It also checks for previously added object's field values and do not allow new object to be added in to object oriented database, if new object has field whose value is same as previously added values for same field(for previously added object) and that filed has assigned unique constraints [10].

Structural change: I have discussed above is, what general change needed to do to add functionality of the constraints in my proposal. In that change, I have appended a new string named [Constraint Name] which is separated with the [Field Value Separator 3]. In all constraint (unique, primary, foreign) case [Field Value Separator 3] is same but [Constraint Name] is different in every constraints. In unique constraint value of [Constraint Name] is "Constraint:unique" so in the unique case [*Field Values*] is as follow,

[*Field Values*] = [Field Name] + [Field Value Separator1] + [Field Type] + [Field Value Separator 2] + [Field Generic Type] + [Field Value Separator 3] + "Constraint:unique"

"Constraint:unique" – this string indicates "this field" contains constraint named unique. And "this field" is the field of class which is represented by [*Field Values*].

Above discussion is all about, what structural change needed to add new functionality of the unique constraints but some logical changes are also needed to implement them in to my demo. And the logical changes are explained following.

Logical change: Add new function to API, may be named "addUniqueConstraint" which takes class object and field name as argument. This function will append the string "Constraint:unique" to the associated [*Field Values*] for the field name which is passed as argument. Now onwards need to check for the uniqueness for the fields that are associated with unique constraints during the insert and update objects. To maintain uniqueness for field need to check all values that are previously added for that field.

Before adding this constraints to the field need to check for uniqueness of that field records (values) which was previously added to my proposed oo database and if duplication found in the previously added records for that field then do not allow user to add this constrain. Inform user that make all value unique for that field before applying unique constraint on it.

Another way to add constraint in to database is to create method inside the class whose object need to store in to object oriented database. Function might have name "addUniqueConastraint". This function returns the String. This return string contains the fields name with comma separation where this unique constraint is needed. At the time of insertion of the object add functionality which will check for this method. And if method available in class associated with that object then execute that method dynamically by using java reflection API [13] and return string of that executed function has the field names with comma separated. After getting all field name above explain technique (appending string and checking for uniqueness) can be applied to implement unique constraint to my demo implementation of the object oriented database.

Challenges: At the time of adding unique constraints one of the major challenges is, to optimize this functionality such that it does not affect on the performance. In short need avoid field uniqueness checking when it is not required.

Primary key constraint:

Primary key constraint is quite similar to the unique key constraint. Only difference between these two constraints is one is allow null value and other does not allow null values to store into database, Primary key does not allow null value while unique key does allow null values [10]. This is mainly used to create identity for the field of class. It is also can be used for the foreign key assignment [10]. To implement this functionality some structural change needed in the class structure file and some logical changes needed in the demo implementation to support this constraint.

Structural change: Above I have already discussed the general structural changes that are needed to add constraints functionality in my proposed object oriented database. By being specific for

the primary key constraints, [Constraint Name] is replaced by "Constraint:primary" so the [*Field Values*] will be:

[*Field Values*] = [Field Name] + [Field Value Separator1] + [Field Type] + [Field Value Separator 2] + [Field Generic Type] + [Field Value Separator 3] + "Constraint: Primary"

"Constraint: Primary" – is the string which indicate that primary constraint is associated with this field.

This is the structural changes needed in my proposed design but there also some logical changes is needed to implement primary key constrains and they are as following.

Logical Change: To add this functionality, some logical functionality needs to be changed. To add primary key constraint one thing can be done and that is, provide new function that takes argument as class object and field name. Class object is used to refer associated class, and field name is the field which needs to be primary key. This function adds "Constraint:primary" string inside the [*Field Values*] for that field.

Now need to change logical functionality for the insert function. At each insertion of new object which contains primary field, need to check uniqueness and not null value for that field. Also need to change logical functionality for the update. At each update of the primary field contain object need to check for the uniqueness and not null values for that field (field that contains primary number).

In other approach to provide primary key only need to create function in the class whose object is going to store object oriented database. This function might be named "addPrimaryKey" which returns the string of field name. Now need to change logic for the class structure insertion function. In that, need to search for the function with name "addPrimaryKey" inside the class whose structure needs to be stored (inside the class structure file). If "addPrimaryKey" function is found then need to execute that function dynamically using java reflection [13] and get the return string of that function. This return string contains the field name. Now need to append extra string with this the field to make them primary (as discussed above). Rest of logical change with insert and update function is same as above.

Challenges: To implement this functionality some challenges need to be faced. The major challenge is to change logic of insert and update such that it doesn't effect on the non constrained class/field performance.

Foreign key constraint:

Foreign key constraint is mainly used to refer one field of the class to the other field of the class. This reference concept is very popular in the relational database systems [9]. Field with foreign key constrains must refer to the other field which contain primary key constraint. Even most popular object oriented database available today does not provide this functionality, in my approach it is possible to provide this functionality with less efforts [19] [20] [22] [23]. To implement this functionality needs some structural changes as well as the logical changes.

Structural change: As propose in general structure change (in beginning of this topic) that is needed to implement constraint functionality, and that is, to add new string [Constraint Name] to

[*Field Values*]. In the case of the foreign key [Constraint Name] should be replace by three more strings concatenated with each other through some separator. And strings are "Constraint:foreign", "ReferenceClassNumber:[class unique number]" and "FleldName: [Field name]"

So structure of the [*Field Values*] becomes as follow:

[*Field Values*] = [Field Name] + [Field Value Separator1] + [Field Type] + [Field Value Separator 2] + [Field Generic Type] + [Field Value Separator 3] + "Constraint:foreign" + [Field Value Separator 4] + "ReferenceClassNumber:[class unique number]" + [Field Value Separator 4] + "FleldName: [Field name]"

"Constraint:foreign" - this string indicates that this string contains the constraints named foreign.

"ReferenceClassNumber:[class unique number]" - this string indicates that this (field whose schema value is assign by [*Field Values*]) field is refer to this (class which has unique number = [class unique number]) class's field.

"FleldName: [Field name]" – this string indicates that this (field whose schema value is assign by [*Field Values*]) field is refer to the field with named [Field name] of class which has unique number [class unique number] in database.

These all are the structural changes needed in my design to add constraint of foreign key. To implement foreign constraints to my demo implementation of the object oriented database so many logical changes are needed. Following are the logical changes that are needs to be done.

Logical Change: To implement foreign key constraint to my demo is different from the above two constraints. This constrains require lots of changes in logic to implement this functionality of foreign constraints.

First thing need to add new function that can be helpful to assign foreign constraint to field. This function takes arguments like object of class, field name, object / name of reference class, and reference field. This function append appropriate values according to above discussion, to the [*Field Values*] String to assign foreign constraint to the given field.

Next step is to change functionality logic for insertion and update. During insertion any [*Field Values*] of the class structure with the string "Constraint: foreign" found then, using the reference class number and reference field name get all the values for the reference field. Now compare current field value with the reference field if match found then insert the value else give error to user that this value is not available in reference filed.

Same logical changes needed for the currently available update functionality. Need to check for reference value for every update of the foreign constrained field.

Challenges: To implement these foreign constraints is the full of challenges. Main challenge is to check for reference field values. Another main challenge is to check for foreign and reference constrains in refactoring functions like rename field, rename class and move fields. To read all

the values of reference class might be time consuming, so another challenge is to make this functionality less time consuming.

Foreign keys field is dependent on the primary key field so when user removes or update primary field data then related action need to perform inside the foreign key. To do that new functionality can be implemented for the foreign and primary key constrains and they are on update cascade and on delete cascade. Following proposal shows how these functionalities can be implemented in my proposed object oriented database system.

On Update and On Delete cascade for Foreign Keys:

On Delete cascade and on update cascade functionality can be provided for above proposed foreign keys constraints in my demo. To implement these functionalities need to change some structural and logical changes.

On delete cascade is functionality to delete all object associated with the class whose field has foreign key constraint to other class's field and that other class' field is removed but it only removes object's whose foreign field's value is same as removed reference field's value.

On update cascade is functionality which update all the foreign constrained field value if associated reference field value is changed and set new value to the objects of foreign constrained field is the same as changed value of reference field.

To implements these functionality in my proposed system need to change some structure and also need some logical changes. Following are the proposal of structural and logical changes that are needed.

Structural change: To add on update and delete cascade functionality need to update above proposed foreign key structure slightly. One new string is need to be updated with [*Field Values*] to find out which cascade is available. Following is the proposed structure.

[*Field Values*] = [Field Name] + [Field Value Separator1] + [Field Type] + [Field Value Separator 2] + [Field Generic Type] + [Field Value Separator 3] + "Constraint:foreign" + [Field Value Separator 4] + "ReferenceClassNumber:[class unique number]" + [Field Value Separator 4] + "FleldName: [Field name] + [Field Value Separator 4] + cascadeOn: [Update/Delete]"

"Constraint:foreign" - this string indicates that this string contains the constraints named foreign.

"ReferenceClassNumber:[class unique number]" - this string indicates that this (field whose schema value is assign by [*Field Values*]) field is refer to this (class which has unique number = [class unique number]) class's field.

"FleldName: [Field name]" – this string indicates that this (field whose schema value is assign by [*Field Values*]) field is refer to the field with named [Field name] of class which has unique number [class unique number] in database.

"cascadeOn: [Update/Delete]" – this string indicate cascade on functionality is activated to the foreign constrained field. [Update/Delete] is "Update" or "Delete" or "UpdateDelete" or

"None". if [Update/Delete] is "Update" then on update cascade is there with reference key. If its "Delete" the on delete cascade is there with foreign and reference key and if "UpdateDelete" is there the both cascade on update and cascade on delete is associated with the foreign key. If [Update/Delete] is "None" then no update delete cascade is associated with foreign key.

To implement this functionality some logical changes are needed on update and delete functionality. They are as follow.

Logical changes: While updating any filed need to check, is this field is primary key? If yes is, then check, is filed is associated with other field (foreign field) as reference field? If yes, then check, is associated foreign key contains the on update cascade? If yes then change all the foreign fields with new updated value whose have same value as current reference field value which is going to be updated.

At the time of delete field need to check is this field is primary key? If yes is, then check, is filed is associated with other field (foreign field) as reference field? If yes, then check, is associated foreign key contains the on delete cascade? If yes then remove the entire objects values of the foreign field contain class where foreign key has same value as referred deleted field.

Multiple constraints:

If more than one constraints associated with the one field of the class then all constrains are separated by comma. Following are the example of the [*Field Values*] where field has constrains unique key and foreign key.

[*Field Values*] = [Field Name] + [Field Value Separator1] + [Field Type] + [Field Value Separator 2] + [Field Generic Type] + [Field Value Separator 3] + "Constraint:unique" + "," + "Constraint:foreign" + [Field Value Separator 4] + "ReferenceClassNumber:[class unique number]" + [Field Value Separator 4] + "FleldName: [Field name] + [Field Value Separator 4] + cascadeOn: [Update/Delete]"

In above example "Constraint:unique" string indicates the Unique constraint and "Constraint:foreign" + [Field Value Separator 4] + "ReferenceClassNumber:[class unique number]" + [Field Value Separator 4] + "FleldName: [Field name] + [Field Value Separator 4] + cascadeOn: [Update/Delete]" indicates the foreign key constraint and comma(",") is the separator to separate both constraint.

Above some proposal shows how to add constraints to my proposed design of object oriented database.

Smart schema changes (automated schema evaluation):

This functionality is provided by the most of the major object oriented databases available today [20], [22]. This Functionality is change the object automatically with the schema changes in the class. Following automated schema change can be provided using my design. This automation is limited for the insert new field, remove fields, change field type, and rearrange fields. Following are some description how automation can be apply.

Insert new field: If user/programmer add new field to the class schema than at the time of next insertion, update, delete or select of that field change in the database done by the automated system. To implement this first step need to do is to find is there any schema changes in the class. And this is done by the checking current schema of the class with the existing schema of the class in the class structure file. If some schema changes found then check any new field is added. And this thing is done by checking each field of the class to the stored field of class in class structure file. After doing this, if founded new field(s) then do(es) the following steps.

Add the field inside the class structure file. Take the default value of that field and add that value to object values file according to class structure. Only above two steps are needed to implement this functionality.

Change field type: To get the field type change of the class automatically first thing need to check for schema change at the insert/ update/ delete and select functions. To get the schema change need to compare current class structure with the class structure which is stored in to class structure file. If some schema change is found then check class' each field name and its type with the class structure file's field name and type. If name is same but type is change then there should be field type change in the class. If field type change found then, check for ability of conversion. Is ability of conversion is found then, change the field type to the structure of class in class structure file.

Ability of conversion functionality checks all the previously added records of the changed field and check that these values are compatible with new type. If even one value is not compatible to the new type this function says this new type is not compatible with new type of that field otherwise this function says yes it is compatible. Automated change is done only if ability of conversion function says yes.

Remove field: Remove field found same way as add field can found only difference in remove field structure of class stored in class structure file has more field then real class fields. In this case need to remove field from the file named class structure. And remove relevant record from the object value file for that class object values.

Rearrange the fields:

As I told in my class structure file that class structure file contains the structure of class schema. And fields are appending to one string and they are separated by a separator. And object value file store the value of the class object field. This values stored in sequence according to sequence of the field structure stored in class structure file. When user wants to read objects of the particular, according to structure of that class stored in the class structure file my API creates object of that class and fill the values to that object which are stored in to the object value file. These two files are synchronized to each other. So when user/programmer rearrange the fields in class and then try to store values in the class so before storing that value need to check for the rearrangement of the fields. If rearrangement found then, to deal with that need to change sequence in both class structure and object values file.

Security to my approach:

Currently my proposal uses three files system which contains all the information of class like, class field structure and class object values. Currently every one can see my file and its data. But I have some proposal to deal with this problem.

Create files in any random format to confuse the users. If user wants to add further security to the system function can be provided, which can implement symmetric encryption to the files [24] [25]. In symmetric encryption admin selects the keys. By creating one function this key is securely store in API. By using that key every new entry of the field can be store encrypted so if someone views the encrypted files is not able to understand the data associated with it. At the time of reading object values can be decode using same key because of symmetric encryption scheme [24] [25]. This is how files can be secure and non readable.

Add log functionality:

Currently my demo is not creating log for every insert, update and delete of the object. So in my demo tracing and recovery option is not available. But logging facilities can be applied to my approach. To add log functionality my proposed approach is as follow.

Add each and everything to the log files. Either user add, update, remove the object of the class or do schema change in class add each entry to the log files. This log file is simple text file, may be named "log.txt". This file may be encrypted because of the security. Following are the proposed format for each function.

Insert:

Fill the entry in log files at the time of insert function calls. This entry should have following format.

"Insert" [Separator] [Class name] [Separator] [Object Value] [Separator] [Time]. Where,

"Insert" - indicates the insert new value.

[Separator] - indicates the separator to distinguish different fields.

[Class name] - indicates the class name whose object was stored.

[Object Value] - indicates the value of the object and this value is same as the object values files.

[Time] - indicates time when insertion occurs.

This log entry is useful to rollback the records.

Update:

Fill the entry in log files at the time of Update function calls. This entry should have following format.

"Update" [Separator] [Class name] [Separator] [Object New Value] [Separator] [Object Old Value] [Separator] [Time]. Where,

“Update” - This string indicates this entry is for update function

[Separator] – This is used to separate different fields.

[Class name] - This string indicates the class name which is associated with the object whose value is being updated

[Object New Value] - This string indicates the new value of the object that replaced the old value after update function called

[Object Old Value] - This string indicates the old values of class which was replaced by the new values on the call of update function.

[Time] - This String indicates at which time this update operation occurs.

Delete:

For each delete function need to add entry into log file. following string shows the suggested format of the delete entry in the file.

“Delete” [Separator] [Class Name] [Separator] [Object Value] [Separator] [Time]

“Delete” - This string indicates that this record is deleted from the object value file.

[Separator] - This is separator is used to separate different Strings

[Class Name] - This String indicates which class object value is going to deleted

[Object Value] - This is the value of the object, which was removed from the object value file.

[Time] - This String indicates the time at which this value removed

Rename class:

At each time of class renaming need to add entry to the log. Following are the proposed entry structure need to be added at time of class renaming.

“Class Rename” [Separator] [New Class Name] [Separator] [Old Class Name] [Separator] [Time]

“Class Rename” - This indicates this record is associated with the class rename.

[Separator] - This is separator is used to separate different Strings

[New Class Name] - This String indicates new class name who replaced old one

[Old Class Name] - This String indicates old class name which was replaced by old field.

[Time] - This string indicates the time when rename of class occurs.

Update class structure (schema change in class):

For each Refactoring function that is associated with schema change need to add entry inside the log file. These refactoring functions do not include class rename. It includes function like rename field, move field, copy field, add field and remove field and much more.

"Structure Update" [Separator] [Class Name] [Separator] [New Structure] [Separator] [Old Structure] [Separator] [Refactoring Type] [Separator] [Time]

"Structure Update" This String indicates this record in log is associated with the Structure updates of the class. This structure updates includes rename field, move field, copy field, add field and remove field.

[Separator] - This String is used to separate the Strings

[Class Name] - This String indicates which class structure has been changed.

[New Structure] - This String is indicates new Structure of the record. This structure String is same as string that is stored in class structure file.

[Old Structure] - This string indicates the old structure of the class. Need to insert old entry to log before delete them.

[Refactoring Type] - This string indicates which kind of schema change is there? i.e. rename field, move field, copy field, add field and remove field

[Time] - This String indicates the time where above events occurred.

Structure new entry to class structure file:

At the time of first entry of the class' object entry occurs in three different files. At that time new entry to Structure file also occurs. Following is proposed format to do entries in log foe new entry in structure file.

"Structure Insert" [Separator] [Class Name] [Separator] [Structure of Class] [Separator] [Time]

"Structure Insert" This string indicate this log record is associated with the Structure of class insert in to class structure file. This entry occurs for each class' first insertion.

[Separator] - This String is used to separate different String Values.

[Class Name] - This String indicates class name for which structure insert occurred.

[Structure of Class] - This String indicates the structure of the class which is similar to String inserted in to class structure file.

[Time] This String indicates at which time this event occurs.

Class new Entry to class record file:

At the time to add new class entry to the class record file following is the propose structure.

"Class Insertion" [Separator] [Class Value] [Separator] [Time]

"Class Insertion" - This string indicates this log entry is associated with the class name insertion inside class record file.

[Separator] - This string is used to separate different strings.

[Class Value] - This string indicates class name with its number.

[Time] - This string indicates the time at which this event occurs.

Above all log entries are useful to get track of record and it is also useful to rollback to previous record. Sometimes it is also useful to get the record of the deleted fields when required. This log entry is also useful to get old structure of the class. This log functionality can come-up with lots of new and useful functionality in my approach. Need to add log entry while adding or removing constraints to or from the field.

6. Results:

As a part of result, I am going to show advantages of my proposed object oriented database compare to existing database. Following are some advantages of my object oriented database.

6.1 Advantages of my proposed design:

The main aim of this project is to remove the direct relationship of objects to its class in object oriented database. Main reason behind this is, by achieving partial isolation between the objects in database and classes, some tool can be provided, which can be useful to synchronize large schema changes in class with its object in database (schema evolution functionality). By proposing new architecture I have achieved partial isolation of the stored object from their class. In demo implementation, I have developed refactoring (schema evolution) tool which can change stored object structure according to their class structure change. Currently available object oriented database somehow managed to provide schema evolution functionality but this functionality is limited [22] [23] [32].

My proposed object oriented database can provide schema evolution technique more easily compare to other object oriented database available in market [23] [34] [44]. Following are some comparison how new design of object oriented database is better than current one if you consider the scenario of the refactoring (Schema changes). In case of other functionality, my proposed design of object oriented database can implement all of them which are available in today's most famous object oriented database.

Today Versant, objectivity, object DB and db4o are major object oriented database in market for java programming language. All of them provide same functionality to deal with the schema changes of class [22] [23] [32] [34] [44]. So, I am going to compare my database with the db4o object oriented database.

6.1.1 Comparison with the db4o:

Db4o provides refactoring API, which are helpful to deal with schema evolution (class schema change) [19] [20]. But it is complicated and limited. If you compare rename functionality of db4o and my proposed database, then it is almost identical to each other. But in my proposed design it should be run faster because to implement rename class functionality, only thing I have to do is

to replace old name with new name in class record file. But in db40, it is more difficult to do that because of the complex architecture (also shown in section 3.1) [22].

To add new fields, remove new fields there are no functions or API provided in the db4o. Only thing is provided in the db4o is automatic refactoring [20]. In db4o user can add new field and remove new field by enabling automated refactoring functionality. But this functionality is not able to add field with user define values it only assigns the default value. And in remove field it will not remove field from the old records until user calls the defragment functions [20]. While in refactoring tool, I have provided various functions to insert field with user define values and with default values. And I am providing function in refactoring tool of my demo implementation, which can remove fields from the previously stored objects. To provide this kind of functions are quite easy because of my proposed architecture of the object oriented database.

Db4o provides refactoring API which is useful to modify hierarchy of the class. But it doesn't support insertion of class in to hierarchy and removal of the class from the inheritance hierarchies [19] [20]. Using my approach, I can provide insertion of class in hierarchy and also can remove class from the hierarchy.

Db4o is not providing foreign and reference key constraints. My proposed object oriented database can provide the foreign key and reference key constraints. These constraints are helpful to provide the relationship between the different fields of the classes.

I have provided new functionality in my proposed object oriented database. Which is not available in the currently available db4o 8.0 version [19] [20]. The functionality is to move fields from one class to other class and copy field from one class to other class. This function can be useful to move field from one class to other class. Main use of this function is to use it in the inheritance (hierarchy).

In terms of query tool, I can provide all the tools similar to the db4o database. In my demo implementation, I have provided the basic functionality that can insert, update and delete object to/from the database. I have provided some selection functions as well which can deal with the selection of data with or without conditions. My design takes more time to fetch data from the database as compared to other object oriented database, but I can improve my performance by loading the entire object at the time of initialization or by using some similar approach. In short I can provide all the functionality which is currently available in the db4o using this approach and this approach help me out to deal with refactoring functionality more smoothly.

Comparison with other object oriented database (Versant, objectivity, objectDB):

Other object oriented database (Versant, objectivity, objectDB) also provides schema evolution functionality same as the db4o [23] [32] [34].All of them are complex architecture and persistent object storage [23] [32] [34]. So above comparison with db4o is applies to other object database as well.

6.2 How it can be useful for big project:

My implementation can be used in big projects as my proposed design of object oriented database works well for refactoring. Basic functionalities are provided in demo implementation and other useful functionality are proposed which will turn my approach to the good object oriented database and it can be replacement of the existing object oriented database.

As this approach provides good refactoring tool, it can be useful in the big projects. At the time of the refactoring of project, it is common to change class schema to improve the functionality and in that case my proposed object oriented database system with the refactoring tool can be handy to deal with it. So using refactoring tool user can change the stored object structure according to the class schema change so there is no lost of the data.

7. Future Improvements:

Following are some points that can be useful to make this approach more useable. This all points are possible to implement using my approach.

Add some more functionality in the refactoring tool: Currently I have implemented limited functionality for the refactoring tool. But in the future more functionality can be added so that it will be easy for the user to change their schema. Currently I have provided only one function to deal with the class hierarchy but in future I can provide other functionality which can deal with the class hierarchy. Some of them may be to add class in between class hierarchy or to remove class form the middle of hierarchy. More functions to change class schema in database, which can be handy.

Indexing functionality: By adding index functionality my approach becomes speedier for selection. Index can be provided by loading all the values inside the files to the object at the time of indexing and then short values of the field on which index is needed. Now next onwards when user selects the data by using some fastest search user will get data faster than before. But update and delete takes longer time than usual because, both functions need to change data into files and from the loaded objects.

Backup and restore: This functionality can be useful to takes backup of all the records and restore all the data when required, or restore all the data to new project.

Transaction Facility: Add transaction functionality to the approached database. To add Transaction functionality I need do some logical change in my given demo.

Client Server functionality: Add client server functionality to my approached object oriented database. This facility can be provided by doing some structural change to my approached system. And it needs some logical changes to demo version of database to implement this functionality.

File system replacement by XML: Managing file system is an headache. In future, XML file or other related files can replace my existing file system so that storage can become more managed [26]. In future this approach can be change to improve efficiency and reliability of the approach.

Conclusion:

To achieve my aim, ‘to change the way how database works’, I have proposed new architecture for the object oriented database which stores object in different way than other object oriented database. My new approach provides partial isolation to the stored object from its class and this property is useful to provide refactoring tool easily. This approach can also provide all the functionalities which are available in current object oriented database.

References:

1. Abraham Silberschatz, Henry f. Korth, S. Sudatshan, (2006), 'Database System Concept', Mc-Grow Hill Publication, Fifth Edition.
2. Robert R. Seban, (1994), 'An Overview of Object-Oriented Design and C++', pp 65-86.
3. Luiz Fernando Capretz, (2003), 'A Brief History of the Object-Oriented Approach', Software Engineering Notes vol 28 no 2, pp 1-10.
4. R.G.G. Cattell and Douglas K. Barry (2000), 'The Object Data Standard-ODMG 3.0. San Francisco', Morgan Kaufmann Publishers.
5. H. Darwenand C. J. Date, (1995), 'The Third Manifesto, In: ACM SIGMOD RECORD', pp. 39-49.
6. Neal Leavit, (2000), 'INDUSTRY TREND - Whatever Happened to Object-Oriented Databases?', IEEE Computer Society Press, pp. 16-18.
7. Malcolm Atkinson, David DeWitt, David Mailer, Klaus Dittrich, Stanley Zdonik,(1992) 'The object oriented database systems manifesto', Morgan Kaufmann Publishers Inc, pp 1-17.
8. Michael Stonebraker, Lawrence A. Rowe, Bruce G. Lindsay, Jim Gray, Michael J. Carey, Michael L. Brodie, Philip A. Bernstein, David Beech, (1990), 'Third generation database system manifesto', ACM, pp. 31-44 .
9. C.J. Date, (1995), 'An Introduction to Database Systems', Addison-Wesley, 7th edition.
10. Ivan Bayross, 'SQL, PL/SQL the programming language of oracle', BPB Publications, 3rd edition.
11. Herbert Schild , (2006),'Java the Complete Reference', McGraw-Hill Professional, Seventh Edition
12. Glen McCluskey, (1998), 'Article on Using Java Reflection', Oracle, available at http://java.sun.com/developer/technicalArticles/ALT/Reflection/index.html [Last Accessed on: 08/09/11].
13. 'Java Platform, Standard Edition 7, API Specification', Oracle, Available at http://download.oracle.com/javase/7/docs/api/index.html [Last Accessed on: 08/09/11]
14. 'Tutorial on What Is an Object?', Oracle, Available at http://download.oracle.com/javase/tutorial/java/concepts/object.html [Last Accessed on: 12/09/11]
15. 'Tutorial on Examining Enums', Oracle, Available at http://download.oracle.com/javase/tutorial/reflect/special/enumMembers.html [Last Accessed on: 13/09/11]
16. 'Tutorial on Getting and Setting Fields with Enum Types', Oracle, Available at http://download.oracle.com/javase/tutorial/reflect/special/enumSetGet.html [Last Accessed on: 13/09/11]

17. 'Tutorial on creating new array', Oracle, Available at http://download.oracle.com/javase/tutorial/reflect/special/arrayInstance.html [Last Accessed on: 13/09/11]
18. Michael L. Horowitz , (1991), 'An Introduction to object oriented database and database system', Carnegie Mellon University
19. 'Tutorial on db40', Versant Corporation, Available at http://developer.db4o.com/Documentation/Reference/db4o-7.12/java/tutorial/ [Last Accessed on: 14/09/11]
20. 'Reference to db40', Versant Corporation, Available at http://developer.db4o.com/Documentation/Reference/db4o-7.12/java/reference/ [Last Accessed on: 14/09/11]
21. Tilmann Zaschke and Moira C. Norrie, (2010), 'Revisiting Schema Evolution in Object Databases in Support of Agile Development', Institute for Information Systems, Switzerland, pp 10-24.
22. 'Product information about db4o', (2006), version 6.0, Available at http://www.db4o.com/about/productinformation/db4o%20Product%20Information%20New%20in%20V6.0.pdf [Last Accessed on: 14/09/11]
23. Dirk Bartels *and Robert* Benso, 'WHITE PAPER of OBJECT PERSISTENCE AND AGILE SOFTWARE DEVELOPMENT', versant, Available at http://www.versant.com/pdf/VSNT_WP_agile.pdf [Last Accessed on: 20/09/11].
24. Nigel Smart, (2002), 'Cryptography, An Introduction' 3rd Edition.
25. Alfred Menezes, Paul van Oorschot, Scott Vanstone, (2001), 'Handbook of Applied Cryptography', Available at http://www.cacr.math.uwaterloo.ca/hac/ [Last Access on: 21/09/11].
26. Erik t Ray, 'Learning XML', O'Reilly, second addition.
27. 'Tutorial on Variable in Java', Oracle, Available at http://download.oracle.com/javase/tutorial/java/nutsandbolts/variables.html [Last Accessed on: 17/09/11].
28. 'Tutorial on primitive data type in Java', Oracle, available at http://download.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html [Last Accessed on: 19/09/11]
29. 'Tutorial on enum type in Java', Oracle, Available at http://download.oracle.com/javase/tutorial/java/javaOO/enum.html [Last Accessed on: 19/09/11]
30. 'Tutorial on methods in Java', Oracle, Available at http://download.oracle.com/javase/tutorial/java/javaOO/methods.html [Last Accessed on: 19/09/11]
31. Dave Minter, Jeff Linwood , 'Beginning Hibernate', Aprees
32. 'manual for the ObjectDB', Available at http://www.objectdb.com/java/jpa [Last Accessed on: 20/09/11]

33. Tatjana Welzer, Shuichiro Yamamoto, Ivan Rozman, (2002), 'Knowledge-based software engineering, Chapter : an automatic method for refactoring java programs', Available at http://books.google.com/books?id=dYRwQYIWqjcC&pg=PA167&dq=refactoring+in+software+engineering&hl=en&ei=rtx4TrPHF4aZ8QO5teSSDQ&sa=X&oi=book_result&ct=result&resnum=3&ved=0CDoQ6AEwAg#v=onepage&q=refactoring%20in%20software%20engineering&f=false [Last Accessed on: 18/09/11]
34. 'feature information about the objectivity (Object Oriented database)', Available at http://www.objectivity.com/pages/objectivity/features.asp [Last Access on: 19/09/11]

35. 'Relational database figure', Available at: http://mwolk.com/blog/relational-database/ [Last Accessed on: 12/05/2011]

36. Jie Lin, Jiankun Yu, Zhiyong Zeng, 'The predicate formulae for object-oriented database schema evolution', IEEE, pp 346-349

37. 'Object oriented database figure', Available at http://www.ukessays.com/essays/computer-science/data-models.php [Last Accessed on: 13/05/11]

38. Martin Fowler, Kent Beck, John Brant, Williams Opdyke, Don Roberts, (2004), 'Refactoring: Improvement the design of exiting code', 14th print, Available at http://books.google.com/books?id=1MsETFPD3I0C&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false [Last Accessed on: 09/09/11]

39. Scott W. Ambler, Pramod J. Sadalage, 'Refactoring Databases: Evolutionary Database Design', Addison Wesley professionals.

40. Dr. Winston Royce, (1970), 'Managing the Development of Large Software Systems', pp. 328-338.

41. The Government of the Hong Kong Special Administrative Region, 'An Introduction to Object Oriented Methodology' available at http://www.ogcio.gov.hk/eng/prodev/download/g52a_pub.pdf [Last Accessed on: 22/09/11]

42. 'Next generation Object standards', available at http://www.odbms.org/odmg/ng.aspx - [Last Accessed on: 20/09/11]

43. 'Object management group', available at http://www.omg.org/ [Last Accessed on: 15/09/11]

44. 'Versant object database datasheet', Versant Corporation, Available at http://www.versant.com/pdf/vod_datasheet.pdf [Last Accessed on: 20/09/11]

45. ‘Object store database information’, Available at http://www.progress.com/realtime/devnet/library/whitepapers/public/objectstore_architecture_introductory.pdf [Last Accessed on: 17/09/11]

46. ‘Object oriented database v/s Object Relational database’, Available at http://www.cs.sfu.ca/cc/354/zaiane/material/notes/Chapter9/node13.html [Last Accessed on: 13/09/11]

47. Jim Paterson, Stefan Edlich, Henrik Hörning, (2006), ‘The definitive guide to db4o’, Apress, Available at

http://books.google.co.uk/books?id=SXG19zl8qF4C&printsec=frontcover&dq=db4o&hl=en&ei=71mDTqSQFc_KsgbQ4426Dg&sa=X&oi=book_result&ct=result&resnum=1&ved=0CDMQ6AEwAA#v=onepage&q&f=false [Last Accessed on: 25/09/11]

Appendix:

As I have discuss in my design part that, I have provide two tools in my demo implementation one of them is query tool and other one is the refactoring tool. In appendix this report provides some brief introduction about the functions available in both tools.
Below all functions of query tools are provided with their description, those all functions of the DBFactory.java.

Function:
public static void save(java.lang.Object o)
Description:
This method saves given object to persistent storage / File Database.
Parameters:
o – This is the object which user wants to store in database
Throws:
java.lang.IllegalAccessException
java.lang.InstantiationException

Function:
static int deleteAllUsingObject (java.lang.Object o)
Description:
This Function is useful to delete all objects of the class, whose referenced object passed as argument.
Parameters:
o – This object refers the class and deletes all objects of that class
Returns:
Number of objects which are deleted from database

Function:
static int deleteObjectsUsingCondition (java.lang.Object o,
java.lang.String[] conditionVariableName,
java.lang.String[] conditionVariableValue)
Description:
This Function deletes object from the database according to conditions.
Parameters:
o – This object refers the class and delete object of that class which satisfy the condition
conditionVariableName – Field name which is going to use for condition
conditionVariableValue – Field value for the condition variables.
Returns:
Number of objects, which are deleted from database

Functions:
static java.util.ArrayList<java.lang.Object> getAllRecordUsingObject(java.lang.Object obj)
Description:

This Function reads the object from the class.
Parameters:
obj – This object is useful to refer class associated with it and reads all object of that class.
Returns:
Array List which contains objects from database

Function:
```
static java.util.ArrayList<java.lang.Object> getRecordByCondition ( java.lang.Object o,
                                                                    java.lang.String variableName,
                                                                    java.lang.String variableValue )
```
Description:
This function returns all objects which specify the condition.
Parameters:
o - This is a reference of the class whose object needs to read
variableName – This is parameter should be filed name which is going to use for the condition
variableValue – This parameter should be field value for the condition
Returns:
It returns all object in array list which satisfy the conditions

Functions:
```
static java.util.ArrayList<java.lang.Object> getRecordByConditions( java.lang.Object o,
                                                                    java.lang.String[] variableNames,
                                                                    java.lang.String[] variableValue )
```
Description:
This function returns objects which fulfill the conditions.
Parameters:
o - This object is useful to take reference of the class
variableNames – This is array of the field names which are going to use for the condition
variableValue – This is the value for the above mentioned field names for condition
Returns:
This function returns the array list of the object which satisfy given condition

Functions:
```
static int updateAllUsingObject( java.lang.Object o,
                                 java.lang.String[] variableNames,
                                 java.lang.String[] variableNewValues )
```
Description:
This Function is useful to update all objects according to the object passed as argument
Parameters:
o – This object is useful to refer class associated with it
variableNames – This represents field names which need to update
variableNewValues – this represent new value of above mentioned fields which replaces the old value for that field.
Returns:
Number of objects that are updated

Functions:
static int updateObjectsUsingCondition(java.lang.Object o,
java.lang.String[] variableNames,
java.lang.String[] variableNewValues,
java.lang.String[] conditionVariableName,
java.lang.String[] conditionVariableValue)
Description:
This Function updates object according to conditions
Parameters:
o – This is useful to refer a class associated with this object
variableNames – this is array of field name which needs to update
variableNewValues – this is value of field which replaces the old value of object
conditionVariableName – This is field names which are going to use for conditions
conditionVariableValue – this is the value for the condition fields.
Returns:
It returns number of object which are updated

Refactoring Tool:

Below all functions of Refactoring tools are provided with their description, those all functions of the RFFactory.java.

Function:
public static java.lang.String getStructureFromDB(java.lang.String className)
Description:
This function returns structure of class which is passed as the argument.
Parameters:
className – This parameter indicates class name
Returns:
It returns structure of the class stored in database

Function:
public static void renameClass(java.lang.String oldName,
java.lang.String newName)
Description:
This function is useful to rename the class
Parameters:
oldName – this is old class name
newName – new name of the class

Functions:
public static void renameFields(java.lang.String className,
java.lang.String[] oldFieldName,

java.lang.String[] newFieldName)

Description:
This Function is useful to change name of the class' field(s)
Parameters:
className – class name in which need to change
fieldoldFieldName – old name of fields (inside above className)
namesnewFieldName – new name of fields (inside above className)

Function:
public static void moveFields(java.lang.String originalClassName,
java.lang.String newClassName,
java.lang.String[] FieldNames)
Description:
This function moves field from original class to new Class it will also transfer its values (if possible)
Parameters:
originalClassName – This is the class who currently holding the fields
newClassName – This is the class where need to move the fields
fieldFieldNames – field names in array which need to move

Functions:
public static void copyField(java.lang.String originalClassName,
java.lang.String newClassName,
java.lang.String[] FieldNames)
Description:
This function copies Fields from one class to other class and its value if possible
Parameters:
originalClassName – class name which contains all fields
newClassName – class name where you want to copy field
fieldsFieldNames – field names in array which user wants to copy

Function:
public static void changeFieldType(java.lang.String className,
java.lang.String[] fields,
java.lang.String[] newTypes)
Description:
This function will changes the Field type according to given new type new type can be int, float double, String and Boolean
Parameters:
className – class name which contains all field
fields – fields names whose data type need to change
newTypes – new types of fields

Function:
public static void deleteFields(java.lang.String className,
java.lang.String[] fieldnames)

Description:
This function is useful to remove fields from the class
Parameters:
className – class which contains the fields
fieldNames – fields name in array which are needs to remove

Function:
```
public static void addFieldsWithDefultValue( java.lang.String className,
                                             java.lang.String[] FieldNames,
                                             java.lang.String[] FieldTypes)
```
Description:
This method is useful to insert new field inside the classs
Parameters:
className – class name where you want to insert new field
FieldNames – fields name in array
FieldTypes – field type for new fields

Function:
```
public static void addFieldsWithValue( java.lang.String className,
                                       java.lang.String[] FieldNames,
                                       java.lang.String[] FieldTypes,
                                       java.lang.String[] values )
```
Description:
This function is add field and assign value to that field
Parameters:
className – class name where you want to insert
fieldFieldNames – fields name in array which user want to insert
FieldTypes – fields type in array which user want to insert (synchronized with filedType)
values –values of the field in array (synchronized with fieldType)

Function:
```
public static void removeSuperClasses ( java.lang.String className )
```
Description:
This function removes All super class reference from subclasses
Parameters:
className – class where you want to remove inheritance

Function:
```
public static java.lang.String getAllClassFromDB()
```
Description:
This function is useful to get all class name which are stored in to database
Returns:
String with all class names.

For the demo I have provide some class which uses above function in my demo_pre package of code. I have given some description in the class which might be useful to know, how to use that?