2002

# Incremental object horizontal fragmentation.

Pinakpani. Dey
*University of Windsor*

# Incremental Object Horizontal Fragmentation

By

Pinakpani Dey

A Thesis
Submitted to the Faculty of Graduate Studies and Research
through the School of Computer Science in
Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

2002

Pinakpani Dey 2002

# Abstract

In intranets, extranets and internet applications, data are by nature complex and distributed over different sites. Object-oriented database management systems meet the requirements of these applications. It offers complex structures, object identity, inheritance between classes and extensibility to capture complex data. A distributed database system partitions large and complex data into smaller pieces and allocates them at different sites to enhance application performance by reducing data communication and replication costs. The design issues of distributed database system require solving several interrelated problems: data fragmentation, allocation and optimization. There are three types of fragmentation – horizontal, vertical and hybrid.

Horizontal fragmentation of a class keeps all attributes and methods of the class but some instance objects in each horizontal fragment. In other words, a horizontal fragment is a subset of class extent or instance objects. Application queries, query access frequencies, instance objects, and object database schema including class composition hierarchies and class inheritance are used as input to generate these fragments. When there are major changes in these input over time, the performance of the distributed object-based system degrades and requires re-fragmentation. The re-fragmentation is started from scratch with static fragmentation approach using all input data (old and changed part).

In this thesis, we propose a new algorithm called Incremental Object Horizontal Fragmentation (IOHF) for distributed object-oriented database systems. This algorithm uses the changed part of input data and previous fragments to define new fragments more quickly, saving system resources and making data at distributed sites more available for network and web access.

**Keywords: Object-oriented databases, Incremental horizontal fragmentation, Distribution**

*To my parents and wife,*

*To my teachers,*

*To my friends,*

...

# Acknowledgements

I would like to take this opportunity to thank the people who help me during my graduate study in University of Windsor.

Thanks to my advisor, Dr. Christie Ezeife, for your help and cooperation that enabled me to complete my degree. Thank you, for your precious consulting time and comprehensive guidance through out my graduate studies and thesis research work.

I would also thank my internal reader, Dr. Xiao Jun Chen and external reader, Dr Nader Zamani. Your comments, questions and suggestions helped me to improve the thesis quality.

I would like to thank, Dr. Arunita Jaekel for chairing my thesis committee.

I would like to thank my parents, wife, brothers and sisters for encouraging me to complete my M. Sc. degree.

Special thanks to my colleagues of AIT Systems Inc., for their cooperation and encouragement, that inspires me to accomplish this degree.

Finally, I would like to thank all my friends for their support, advice and help during my M. Sc. Study.

# Table of Contents

# List of Figures

# Chapter 1: Introduction

In the era of information technology, database systems have become an essential component of everyday life in modern society. In the course of a day, most of us encounter several activities that involve some interaction with a database [EN00]. Most of these data are by nature complex and can not be mapped into a two dimensional table in a traditional relational database management system (RDBMS) - for example, data for engineering design and manufacturing (e.g., CAD/CAM and CIM), scientific experiments, telecommunications, geographic information systems, and multimedia. Traditional RDBMS is not adequate to support these types of applications that have much more complex kinds of data. Hence, the database researchers want to develop a database management system that can manage complex data. There are two different trends for developing database management system for complex data [KIM90]. These are:

(1) *The revolutionary approach:* Here, database research is based on a new data model, the object-oriented model [Ram98][KG94], which leads to object-oriented database management system (OODBMS). For example, Jasmine, Gemstone, O2, Object Store, Objectivity/DB and Versant ODBMS are some of the existing object-oriented databases.

(2) *The evolutionary approach:* In this research and development work, the conventional relational model is taken as a platform for extensions and adaptations to produce object relational database management system (ORDBMS). For example, Oracle 9i, Universal Server (Illustra), Universal Database (DB/2 Extenders), UniSQL/X, OSMOS and JDataStore are some of the existing object relational databases.

With increasing availability of enhanced network and data communication protocol, from satellite and cellular communications to the standardization of protocols like ethernet, TCP/IP and internet, distributed database (DDB) technology needs to merge database, network and data communication technology. A distributed database system partitions large and complex data into smaller pieces and allocates them at different sites to enhance application performance by reducing data communication and replication costs. Electronic commerce over the internet, multimedia applications, such as news-on-demand or medical imaging, and manufacturing control systems are all examples of distributed systems.

## 1.1 The revolutionary approach – Object-Oriented Database Systems

An object-oriented system is a collection of objects classified into a finite number of classes. Objects that have the same attributes and methods belong to the same class. This model is more powerful and complex than the relational model. For example, to store multimedia information, databases must store various types of multimedia objects, such as video, audio, images, graphics and documents.

A data model is a logical organization of real world objects (entities), constraints on them, and relationships among objects. A data model that captures object-oriented concepts is an object oriented data model. An object-oriented database is a collection of objects. The behavior, state and the relationships among objects are defined in accordance with an object-oriented data model. "An object-oriented database system is a database system, which allows the definition and manipulation of an object-oriented database" [KIM90]. An object-oriented database can also be defined as a collection of encapsulated objects.

A class C is represented as an order relation such as C = (K, A, M, I) [EB95], where K is a class identifier, A is a set of attributes, M is a set of methods for accessing these attributes and I is a set of instances of objects. Object-oriented database also supports inheritance. For example, class B is a subclass of A, if class B can inherit all attributes and methods of its superclass A. An object can be a simple or complex object and can establish is-part-of relationship with other objects. A group of experts in object-oriented DBMSs published the object-oriented database system manifesto in 1989 as a standard [ABDDMZ89]. A standard OODBMS should include complex objects, object identity, encapsulation, classes, inheritance, polymorphism, computationally completeness, extensibility, data persistence, very large databases, concurrency, recovery, and ad hoc queries features [Rao94].

The first step in defining an object-oriented data model [WT96] is to simply observe and record the objects in the domain. For example, formulate a description of the solution and identify the candidates for being the objects in the data model. Then, identify the characteristics of these objects, which are the object attributes. Examining the logical dependencies among the objects identifies different kinds of association. Analyzing the system decomposition into

subparts can identify the object relationships. Examining the responses of objects identifies different kinds of methods. Finally, the classifications of objects into an inheritance structure can figure out common characteristics and behaviors between objects. These steps are completed in an iterative fashion until the complete data model has been defined.

According to research in [BF95a] [RBKW91] [Ru94] and [CM94] the following advantages of OODB system are presented.

- Providing direct modeling of complex and nested objects: For example, a DEPARTMENT object may have five attributes DNAME, DNUMBER, MGR, LOCATIONS, and EMPLOYEES. Here, DNAME and DNUMBER are basic data type (string). MGR, LOCATIONS and EMPLOYEES are complex data type (class).
- Organizing classes (types) into inheritance hierarchies: For example, PROFESSOR and STUDENT classes can be inherited from the same class PERSON.
- Supporting high-level declarative object query languages (OQL): This is an important feature, which has been retained from the relational model to reduce application development times. For example, to retrieve the names of all professors whose salaries are greater than $60,000 can be written as "select p.NAME from p in PROFESSOR where p.SALARY > 60000".

## 1.2 The evolutionary approach – Object Relational Databases Systems

To support navigation, security, rollback, backup, recovery and data integrity, a new database class was born called object relational database management systems (ORDBMSs). "Object relational DBMSs are relational in nature because they support SQL; they are object-oriented in nature because they support complex data. In essence, they combine SQL from the relational world and the modeling primitives from the object world" [SM96].

The object relational system is the evolutionary method of RDBMS, but there are various approaches for extending relational DBMS with object-oriented concepts. The basic idea is an extension of the "flat" relational data model with constructions that allows for definition and manipulation of complex objects [Sz96]. In [SM96], Stonebraker and Moore, describe object relational database management systems (ORDBMS) having at least the following set of features

as an extension to relational database systems (RDBS): user defined objects, complex objects, inheritance, and rules. There are two kinds of data types in an ORDBS - atomic and structured data types. Atomic data types are built in simple types e.g., integer, string etc. Structured data types are called complex objects or complex types and user can define these types. The functions can also be defined on both atomic and complex types. Functions can be defined in several languages e.g., SQL, C, Java, but this depends on the support from the DBMS. Multiple inheritances are also supported by the systems.

## 1.3 Distributed Object-Oriented Database Systems

With advances in distributed processing and distributed computing that occurred in the operating systems arena, the database research community did considerable work to address the issues of data distribution, distributed query, transaction processing, distributed database and metadata management. They also developed many research prototypes [EN00].

A distributed database is a collection of multiple logically interrelated databases, distributed over a computer network, including intranet and internet. A distributed object based system is a collection of local object based systems that are distributed among different local sites and interconnected by a communication network [EB98]. In distributed object based system (DOBS), class is the entity of distribution [EB95]. A distributed database management system (distributed DBMS) is then defined as the software systems that permit the management of distributed database systems (DDBS) and makes the distribution transparent to its users [RZ95] [OV99]. The DDBS technology should recognize the following four fundamental premises [OV99].

(1) *Transparent management of distributed and replicated data:* Transparency means that the user of the distributed database is shielded from details of data distribution. For example, user at site A does not need to know that the data he needs in his query is at site C.

(2) *Reliability through distributed transactions:* Distributed DBMSs are intended to improve reliability since they have replicated components and thereby eliminate single points of failure. For example, faculty, staff and student are three partitions of a university database. These partitions are allocated into three sites A, B and C. The partition student

is replicated into two different sites A and B. If site A fails, then distributed systems can retrieve data from site B.

(3) *Improved performance:* By fragmenting the conceptual database and utilizing parallel execution, distributed system improves query performance. For example, from (2) above, a query, group graduate students according to their supervisors, retrieves data from both student and faculty partitions. Distributed system partitions the query and executes them in parallel in the student and faculty partitions.

(4) *Easier system expansion:* In a distributed environment, it is much easier to accommodate increasing database sizes. For example, from above (2), as the university database is partitioned and allocated into three separate sites, each partition utilizes different hardware resources. As a result, when the database size increases, the distributed system can utilize resources from all three sites.

Moreover, distributed databases provide function for keeping track of data, distributed query processing, distributed transaction management, replicated data management, distributed database recovery, security, and distributed directory (catalog) management functions [EN00]. Distributed systems are economical, because multiple processing elements and resources are used optimally. Hence, a distributed system divides data and program across different sites efficiently.

There are two alternative methods of distribution - (1) partitioned and (2) replicated. In the partitioned scheme, the database is divided into a number of disjoint partitions, each of which is placed at a different site. Replicated designs can be either fully replicated where the entire database is stored at each site, or partially replicated where each partition of the database is stored at more than one site, but not at all of the sites. So, in case of partitioning, again there are two fundamental design issues.

(1) Fragmentation - the separation of a database into partitions called fragments. For example, the university database is partitioned into student, faculty and staff.

(2) Allocation - the optimum distribution of fragments over different sites. For example, student, faculty and staff partitioned databases are allocated to site A, site B and site C respectively.

There are two possible approaches for distributed systems – top-down and bottom-up [OV99] [EB95]. Top-down approach is useful when we start to design a system from scratch. On the other hand, bottom-up is used to reconstruct a database from different existing databases. For the former one, global conceptual schema (GCS) and access patterns are input to the distribution design process and output will be local conceptual schema (LCS), where GCS describes combined logical organization of data from all sites and LCS describes logical organization of data at each site. Bottom-up approach uses LCS to create GCS. According to [EB95] and [OV99], the top-down approach for distributed system is shown in figure 1.3.1. Here, GCS, access information, and external schema definitions (queries) are used as input to the distribution process. Then, distribution process creates $LCS_i$, where i = 1 to n. These LCS are then used as input to the allocation process. Allocation process generates local internal schema $LIS_j$, where j = 1 to m. LIS describes the physical data organization on a site (machine).

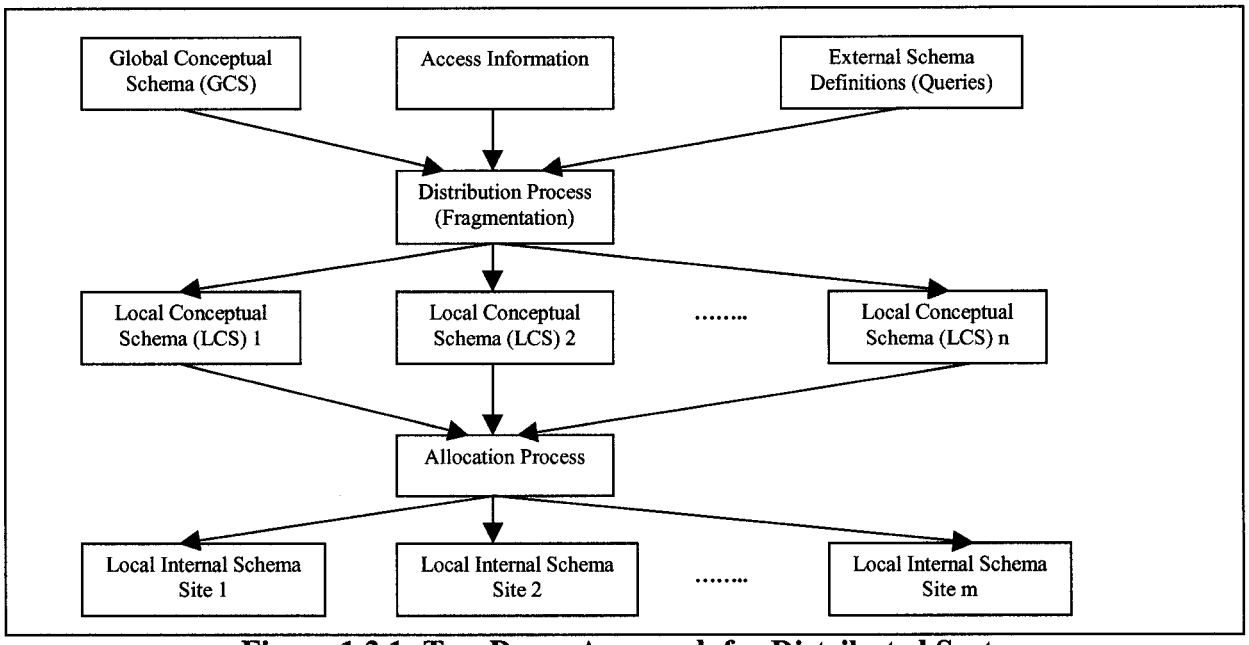


**Figure 1.3.1: Top-Down Approach for Distributed System**

In this thesis, we concentrate on the distribution process to generate local conceptual schema (LCS) from global conceptual schema (GCS) by getting information from access information and external schema definition. Data fragmentation is the process of clustering data from a class, by grouping relevant attributes and objects accessed by an application, into class fragments [BKS97] [EB95]. The advantages of data fragmentation are:

1. It enhances performance of applications, because it reduces the amount of irrelevant data to be accessed and transferred among different sites in a distributed system [BKS97].
2. It decomposes a class into class fragments and permits concurrent processing. As a result, a query can access fragments of the same class simultaneously [LN96].
3. It reduces the amount of data transfer during data migration.
4. It replicates fragments rather than replicate the entire class.

There are three fundamental types of fragmentation [EB95] [KMN93]. These are:

a) ***Horizontal fragmentation*** of a class, which contains all attributes and methods of the class but only some instance objects [EB95]. So, a horizontal fragment of class C is $C_h = (K, A, M, I')$, where $(I' \subseteq I)$. For example, a class employee having simple attributes id, name, age and salaries that range from \$30,000 to \$100,000, has ten instances. A horizontal fragmentation can provide two fragments $F_1$ and $F_2$, where $F_1$ contains the three instances with salaries greater than \$80,000 and $F_2$ contains the seven instances with salaries less than or equal to \$80,000.

b) ***Vertical fragmentation*** of a class, which contains subsets of attributes and methods but all instance objects of the class [EB95]. So, a vertical fragment of class C is $C^v = (K, A', M', I)$, where $(M' \subseteq M)$ and $(A' \subseteq A)$. For example, the ten instances of class employee in (a) above, could be partitioned vertically to create three fragments $F_1$(id, name), $F_2$(id, age) and $F_3$(id, salary) and each of them has all instances.

c) ***Hybrid fragmentation*** of a class, which contains subsets of attributes, methods and instances of the class [EB95]. A hybrid fragment of class C is $C_h^v = (K, A', M', I')$, where $(M' \subseteq M)$, $(A' \subseteq A)$ and $(I' \subseteq I)$. For example, two of the hybrid fragments from the same example above are $F_1$(id, name) for salary <= \$80,000 and $F_2$(id, name) for salary > \$80,000.

## 1.4 Thesis Problem and Contributions

In [EB95], Ezeife and Barker, introduce the algorithm for object horizontal fragmentation. This algorithm uses global conceptual schema (GCS) as an input, which is from static analysis. Major changes such as add new applications, delete existing objects, and add new classes in a domain will cause a redistribution of the system from scratch. In [EZ99a], Ezeife and Zheng, introduce an intelligent dynamic algorithm for object horizontal fragmentation. Though, the algorithm itself can determine the necessity for redistribution, it still fragments the system from scratch. Irrespective of the method used for fragmentation, if we need to regenerate from scratch then it will not only increase the system overhead but also reduce its efficiency.

Mathematically, increment means the value of a variable increases by another value. For example, increment a variable x by 2 means that, new value of x will be x + 2. If the initial value of x is 5 then new value will be 7. There is an antonym of this word called decrement, which is used to decrease the value of a variable. Database has a different concept of increment. Increment means, changes of a system (relational or object-oriented) by both insertion and deletion of records or objects. For example, a view $V_1 = R_1 \otimes R_2$. Here, $\otimes$ means relational join. After adding a tuple $t_1$ in $R_1$, view will be $V_1 = V_1 \cup (t_1 \otimes R_2)$. In this example view $V_1$ is updated incrementally due to the change of $R_1$ with tuple $t_1$. In distributed object based systems, there are several possibilities for changes in the existing system. These are adding new applications, deleting existing applications, moving of applications from one site to another, changing access frequencies of applications, adding new classes, deleting classes, adding instances, deleting instances, and moving instances from one site to another. As a result, in distributed object based system, increment means modify existing fragments depending on changes. For example, assuming a horizontal fragment $F_1$ has $I_1$, $I_2$ and $I_3$ instances i.e., $F_1$:{$I_1$, $I_2$, $I_3$}. If we add a new instance $I_5$ into the system and find that $I_5$ should belong to horizontal fragment $F_1$, then $F_1 = F_1 \cup I_5$ i.e., $F_1$:{$I_1$, $I_2$, $I_3$, $I_5$}. So, without re-fragmenting all instances $I_1$, $I_2$, $I_3$, and $I_4$, we can modify existing horizontal fragment $F_1$ incrementally. Again, assuming a site $S_1$ has a set of fragments $F_1$, $F_2$ and $F_3$, i.e., $S_1${$F_1$, $F_2$, $F_3$}. If there is a new fragment $F_4$ and find that $F_4$ belongs to site $S_1$, then $S_1 = S_1 \cup \{F_4\}$ i.e., $S_1$:{$F_1$, $F_2$, $F_3$, $F_4$}.

In essence, the thesis problem is to develop an algorithm to modify fragments (e.g., $F_1$) due to changes (e.g., addition of instance $I_5$) in existing object based systems. This thesis

proposes a new algorithm that performs incremental horizontal fragmentation of class objects by using the changes in input data with the previous fragments to obtain new sets of best fragments. The proposed incremental horizontal fragmentation scheme, utilizes the technique of object-oriented horizontal fragmentation (OOHF) [EB95] algorithm and the principles of object and fragment affinities to undo or/and redo the effects on fragments of an addition, deletion or change of an input to the fragmentation process. This approach cuts down the use of computer resources, which are dedicated for re-fragmentation processes. It also makes the process of applying object re-fragmentation easier to dynamic distribution environment and the web.

## 1.5 Outline of the Thesis Proposal Document

The remainder of this thesis is structured as follows. Chapter 2 looks at related previous work on horizontal fragmentation of distributed object based systems. Chapter 3 proposes algorithms for incremental object horizontal fragmentation (IOHF) for distributed object based systems. Also, we give an example to show how the algorithms work in comparison with re-fragmentation from scratch. Chapter 4 presents performance analysis between proposed and existing algorithms. Chapter 5 discusses conclusions, future work and time lines.

# Chapter 2: Previous/Related Work

This chapter reviews related previous works on horizontal fragmentation of distributed object-oriented databases as well as some existing real world applications of OODB. Section 2.1 shows some definitions. Section 2.2 reviews earlier work on horizontal fragmentation. Section 2.3 reviews work on performance evaluation of distributed system. Section 2.4 reviews some existing real world application of OODB.

## 2.1 Definitions

Some definitions and two affinity rules are needed for object horizontal fragmentation algorithm. These are:

**Definition 2.1** *Access frequency acco(I)* of an instance object (*I*) is the sum of the access frequencies of all the application queries $Q_i$ accessing the object. Thus, $acco(I) = \sum_i^q acc(q_i, I)$ for all q application accessing the object.

**Definition 2.2** *Relevant accesses RelAcc($F_i$, $F_j$)* measures the amount of access made to local objects of fragments $F_i$ and $F_j$. Thus, Relevant accesses ($F_i$, $F_j$) is the sum of access frequencies $acco(I_i)$ for all $I_i \in (F_i \cap F_j)$ i.e., $RelAcc(F_i, F_j) = \sum_{Ii} acco(I_i)$ for all $I_i \in (F_i \cap F_j)$. For example, if $F_1$:{$I_1$, $I_2$, $I_5$} and $F_2$:{$I_1$, $I_4$, $I_5$} are two fragments, then $RelAcc(F_1, F_2) = acco(I_1) + acco(I_5)$.

**Definition 2.3** *Irrelevant access IRelAcc($F_i$, $F_j$)* are those accesses made to instance objects which are in fragment, $F_i$, or fragment, $F_j$, but not in both. Irrelevant access ($F_i$, $F_j$) is the sum of access frequency for all $I_i \in ((F_i \cup F_j) - (F_i \cap F_j))$ i.e., $IRelAcc(F_i, F_j) = \sum_{Ii} acco(I_i)$ for all $I_i \in ((F_i \cup F_j) - (F_i \cap F_j))$. For example, if $F_1$:{$I_1$, $I_2$, $I_5$} and $F_2$:{$I_1$, $I_4$, $I_5$} are two fragments, then $IRelAcc(F_1, F_2) = acco(I_2) + acco(I_4)$.

**Definition 2.4** *Affinity between two Fragments $F_i$ and $F_j$ of the same class aff($F_i$, $F_j$)* is measured as the count of how frequently objects of these two fragments are needed together by applications. Thus, aff($F_i$, $F_j$) = Relevant access($F_i$, $F_j$) – Irrelevant access ($F_i$, $F_j$). For example, if $F_1$:{$I_1$, $I_2$, $I_5$} and $F_2$:{$I_1$, $I_4$, $I_5$} are two fragments, then $aff(F_1, F_2) = (acco(I_1) + acco(I_5)) - (acco(I_2) + acco(I_4))$.

**Definition 2.5** *The Object-Affinity affo($I_i$, $I_j$),* measures the affinity between two objects $I_i$ and $I_j$ as the sum of the access frequencies of the two objects for all queries (q) that access both objects. Thus, affo($I_i$, $I_j$) is the sum of access frequencies for all q accessing both $I_i$ and $I_j$. For example, if

$I_1$, $I_2$ and $I_3$ are instances of a class, then affo($I_1$, $I_2$) = acco($I_1$) + acco($I_2$), affo($I_1$, $I_3$) = acco($I_1$) + acco($I_3$) and affo($I_2$, $I_3$) = acco($I_2$) + acco($I_3$).

**Definition 2.6** *Object-fragment affinity, ofaff($I_i$, F)* is the sum of object affinities between the object $I_i$ and all objects of the fragment F. Thus, ofaff($I_i$, F) is the sum of affo($I_i$, $I_j$), for all $I_j$ in F, where $j \neq i$. For example, if $F_1$:{$I_1$, $I_2$, $I_5$} is a fragment of a class, then ofaff($I_1$, $F_1$) = affo($I_1$, $I_2$) + affo($I_1$, $I_5$) = acco($I_1$) + acco($I_2$) + acco($I_1$) + acco($I_5$).

**Affinity Rule 2.1** Select the primary fragment that maximizes the affinity measure aff($F_d$, $F_i^p$), where $F_d$ is the derived fragment and $F_i^p$ is a primary fragment in the class. For example, assuming $F_1^p$, $F_2^p$ and $F_3^p$ are primary fragments of a class. $F_d$ is a derived fragment of the class. Again, assuming that aff($F_d$, $F_1^p$) = 30, aff($F_d$, $F_2^p$) = 40 and aff($F_d$, $F_3^p$) = 20. Here, aff($F_d$, $F_2^p$) has maximum affinity measure. As a result, this rule selects primary fragments $F_2^p$.

**Affinity Rule 2.2** The fragment $F_j^h$ that maximizes the function ofaff($I_i$, $F_j^h$) is where $I_i$ is placed. For example, assuming $F_1$:{$I_1$, $I_2$, $I_5$} and $F_2$:{$I_1$, $I_4$, $I_3$} are two fragments of a class. Object-fragment affinity of instance $I_1$ with fragments $F_1$ and $F_2$ are ofaff($I_1$, $F_1$) = 10 and ofaff($I_1$, $F_2$) = 20. Here, $I_1$ has maximum affinity of 20 with fragment $F_2$. So, according to this rule, $I_1$ is placed in the fragment $F_2$.

## 2.2 Horizontal Fragmentation

Horizontal fragmentation (HF) is a process for reducing the number of disk access made during execution of a query by reducing the number of irrelevant objects accessed. Based on queries and object based schema, there are two versions of horizontal fragmentation – primary and derived. Primary horizontal fragmentation of a class is performed, based on predicates of queries accessing this class. Derived horizontal fragmentation of a class is performed based on a horizontal fragmentation of another class [EB95][BKS97]. Derived horizontal fragmentation may occur in the following three methods [EB95]

- Partitioning of a class arising from the fragmentation of its subclasses, which includes both primary and derived fragments of subclass.
- Fragmentation of a class arising from the fragmentation of some complex attribute (part-of) of the class.
- Fragmentation of a class arising from the fragmentation of some complex methods.

In [EB95], Ezeife and Barker, introduce horizontal fragmentation algorithms for four types of class models which include (i) class consisting of simple attributes and simple methods, (ii) class consisting of complex attributes and simple methods, (iii) class consisting of simple attributes and complex methods, and (iv) class consisting of complex attributes and complex methods. The inputs to these algorithms are global conceptual schema (GCS) and the access pattern information. The output of these algorithms is a set of local conceptual schemas (LCS).

To facilitate the brief description of object horizontal fragmentation algorithm (OOHF), we use classes Company, AutoCompany, TruckCompany, Vehicle, Automobile and Truck as a

**Company {CID, Name, Country, GetName, GetCountry}**
$I_1$ {CID1, GM, Canada}
$I_2$ {CID2, GM, USA}
$I_3$ {CID3, FORD, USA}

**AutoCompany {CID, CAID, NoEmp, Asset, GetEmp, GetAsset}**
$I_1$ {Company Pointer1, CAID1, 2000, $5,000,000}
$I_2$ {Company Pointer2, CAID2, 5000, $10,000,000}
$I_3$ {Company Pointer2, CAID3, 1000, $1,000,000}
$I_4$ {Company Pointer3, CAID4, 7000, $10,000,000}

**TruckCompany {CID, CTID, NoEmp, Asset, GetEmp, GetAsset}**
$I_1$ {Company Pointer2, CTID1, 3000, $1,000,000}

**Vehicle{VID, Weight, CID, GetWeight}**
$I_1$ {VID1, 300, Company Pointer 1}
$I_2$ {VID2, 400, Company Pointer 1}
$I_3$ {VID3, 300, Company Pointer 2}
$I_4$ {VID4, 500, Company Pointer 2}
$I_5$ {VID5, 300, Company Pointer 3}
$I_6$ {VID6, 400, Company Pointer 3}

**Automobile {VID, VAID, NoDoor, Type, GetDoor, GetType}**
$I_1$ {Vehicle Pointer1, VAID1, 4, CAR}
$I_2$ {Vehicle Pointer1, VAID2, 4, MINI VAN}
$I_3$ {Vehicle Pointer2, VAID3, 2, SPORTS CAR}
$I_4$ {Vehicle Pointer3, VAID4, 4, CAR}
$I_5$ {Vehicle Pointer5, VAID5, 4, MINI VAN}
$I_6$ {Vehicle Pointer6, VAID6, 2, SPORTS CAR}

**Truck {VID, VTID, Color, GetColor}**
$I_1$ {Vehicle Pointer4, VTID1, White}

**Figure 2.2.1: Sample Object Database Schema**

sample data, shown in figure 2.2.1. Each class has its own attributes, methods and instances. For example, ID (CID), Name and Country are attributes of class Company. GetName and GetCountry are methods of the class Company. $I_1$, $I_2$ and $I_3$ are instances of the class Company.

Again, each instance of a subclass contains pointer to instance of its super class. For example, instance $I_3$ of class AutoCompnay contains pointer to instance $I_2$ of its super class, Company.

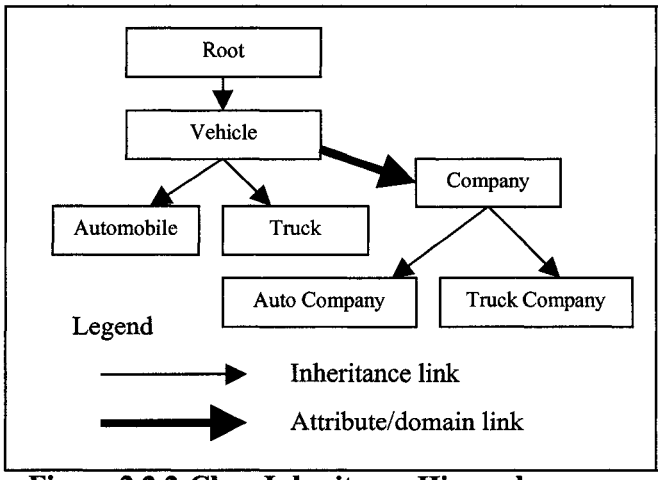


**Figure 2.2.2:Class Inheritance Hierarchy**

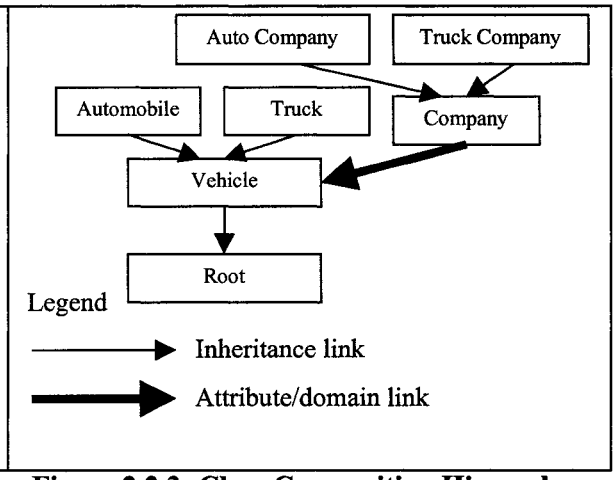


**Figure 2.2.3: Class Composition Hierarchy**

Figure 2.2.2 shows the class inheritance hierarchy, where class AutoComany and class TruckComapny are subclasses of class Company. Class Automobile and class Truck are subclasses of class Vehicle. Again, class Company is a complex attribute of class Vehicle. The classes of the sample database are fragmented based on the queries from figure 2.2.4.

**Query 1:** This query groups objects of class *Company* according to their name. Query requires method GetName of class *Company* and predicates from this query are $P_1$:Name = GM and $P_2$:Name = FORD.

**Query 2:** This query groups objects of class *AutoCompany* according to their asset, where Asset > \$5,000,000 and Asset ≤ \$50,000,000. Query requires method GetAsset of class *AutoCompany* and predicates from this query are $P_1$:Asset > \$5,000,000 and $P_2$:Asset ≤ \$5,000,000.

**Query 3:** This query groups objects of class *Automobile* according to their types. Query requires GetType method of class *Automobile* and predicates from this query are $P_1$:Type = "Car", $P_2$:Type = "Mini van" and $P_3$:Type = "Sports car".

**Figure 2.2.4: Application Queries**

This algorithm generates dependency graph for all classes of the database from the given class inheritance hierarchy such that dependency graph captures the inheritance, method and attribute links between classes. For example, as class AutoCompany and class TruckCompany are subclasses of class Company, algorithm inserts arcs from class AutoComapny (owner class)

to class Company (member class) and from class TruckCompany (owner class) to class Company (member class) as inheritance composition hierarchy (figure 2.2.3). Again, it inserts an arc from class Company to class Vehicle as attribute composition hierarchy.

Now, it generates simple predicates $P_j$: $A_i$ $\theta$ Value, for all classes from queries, where $A_i$ is an attribute of a class, $\theta \in \{=, <, \neq, \leq, >, \geq\}$ and Value is chosen from the domain of $A_i$. For example, simple predicates for all classes from queries are,

$P_{Company}$: {Name = "GM", Name = "FORD"}

$P_{AutoCompany}$: {Asset > \$5,000,000, Asset ≤ \$5,000,000}

$P_{TruckCompany}$: {}

$P_{Vehicle}$: {}

$P_{Automobile}$: {Type = "Car" Type = "Mini van" Type = "Sports car"}

$P_{Truck}$: {}

Then, it generates set of complete minterm predicates and primary fragments of each class. Minterm predicate is the conjunction of simple predicate either in its natural form or its negated form. The minterms and primary fragments are,

**Class Company**

Minterms: $MM_1$: Name = "GM" ∧ Name ≠ "FORD"

$MM_2$: Name ≠ "GM" ∧ Name = "FORD"

Fragments: $f_1^p$: $\{I_1, I_2\}$ $f_2^p$: $\{I_3\}$

**Class AutoCompany**

Minterms: $MM_1$: Asset > 5,000,000

$MM_2$: Asset ≤ 5,000,000

Fragments: $f_1^p$: $\{I_2, I_4\}$ $f_2^p$: $\{I_1, I_3\}$

**Class Automobile**

Minterms: $MM_1$: Type = "Car" ∧ Type ≠ "Mini van" ∧ Type ≠ "Sports car"

$MM_2$: Type ≠ "Car" ∧ Type = "Mini van" ∧ Type ≠ "Sports car"

$MM_3$: Type ≠ "Car" ∧ Type ≠ "Mini van" ∧ Type = "Sports car"

Fragments: $f_1^p$: $\{I_1, I_4\}$ $f_2^p$: $\{I_2, I_5\}$ $f_3^p$: $\{I_3, I_6\}$

As class TruckCompany, class Truck and class Vehicle do not have any predicates, so there is no primary fragment for these classes.

Now, derived fragments are generated on member classes. Derived fragmentation is defined on member classes of links, according to both primary and derived fragments of owner classes. For example, owner class AutoCompany of member class Company has two primary fragments and no derived fragment. So, class Company has two derived fragments, these are $f_1^d$:{$I_2$, $I_3$} and $f_2^d$:{$I_1$, $I_2$}. Similarly, derived fragments of class Vehicle are $f_1^d$:{$I_1$, $I_3$}, $f_2^d$:{$I_1$, $I_5$} and $f_3^d$:{$I_2$, $I_6$}.

Next, it integrates primary and derived fragments to generate the horizontal fragments. To integrate, it applies affinity rule 2.1 among primary and derived fragments of each class. For each derived fragment $F^d$, affinity rule 2.1 selects a primary fragment that maximizes affinity measure aff($F^d$, $F_i^p$). Then, it merges the derived fragments with the primary fragments. If a class does not have any derived fragments, then the primary fragments will be horizontal fragments. For example, as class AutoCompany does not have any derived fragments, so the horizontal fragments are $f_1^h$:{$I_2$, $I_4$} and $f_2^h$:{$I_1$, $I_3$}. Similarly, the horizontal fragments of class Automobile are $f_1^h$:{$I_1$, $I_4$}, $f_2^h$:{$I_2$, $I_5$} and $f_3^h$:{$I_3$, $I_6$}. To calculate maximum affinity, we need the access frequencies of instances based on queries in each class. Based on queries of figure 2.2.4, the access frequencies of class Company and class Vehicle are

| | | | |
|---|---|---|---|
| Class Company: | acco($I_1$) = 5 | acco($I_2$) = 5 | acco($I_3$) = 10 |
| Class Vehicle: | acco($I_1$) = 10 | acco($I_2$) = 5 | acco($I_3$) = 5 |
| | acco($I_4$) = 5 | acco($I_5$) = 10 | acco($I_6$) = 5 |

Applying affinity rule 2.1, to fragments of class Company, $f_1^d$ has maximum affinity with $f_2^p$ of 5. So, after merging $f_1^d$ with $f_2^p$, we get $f_1^h$:{$I_2$, $I_3$}. Again, $f_2^d$ has maximum affinity with $f_1^p$ of 10. So, after merging $f_2^d$ with $f_1^p$, we get $f_2^h$:{$I_1$, $I_2$}. To create disjoint horizontal fragments, it applies affinity rule 2.2 among the fragments. If an instance is overlapping in different fragments, then affinity rule 2.2 selects a fragment that maximizes object fragment affinity measure ofaff(I, $F_j$) and places the instance into that fragment. Here, $I_2$ is overlapping in both fragments. Applying affinity rule 2.2, $I_2$ is placed in $f_1^h$ with maximum affinity of 15. Hence, the horizontal fragments of class Company are $f_1^h$:{$I_2$, $I_3$} and $f_2^h$:{$I_1$}.

Again, as class Vehicle does not have any primary fragments, so the derived fragments are the horizontal fragments. Hence, horizontal fragments are $f_1^h$:{$I_1$, $I_3$}, $f_2^h$:{$I_1$, $I_5$} and $f_3^h$:{$I_2$, $I_6$}. Here, $I_1$ is overlapping in two fragments $f_1^h$ and $f_2^h$. Applying affinity rule 2.2, $I_1$ is placed in

$f_2^h$ with maximum affinity of 20. Now, the horizontal fragments are $f_1^h$:{$I_3$}, $f_2^h$:{$I_1$, $I_5$} and $f_3^h$:{$I_2$, $I_6$}.

Now, for complex attribute, it considers the above horizontal fragments of each class as primary fragments. As the class Company is a complex attribute of class Vehicle, then the derived fragments of class Vehicle are $f_1^d$:{$I_3$, $I_4$, $I_5$, $I_6$} and $f_2^d$:{$I_1$, $I_2$}. Now, it integrates the derived fragments with primary fragments. Applying affinity rule 2.1, $f_1^d$ has maximum affinity with $f_2^h$ of –15 and $f_2^d$ has maximum affinity with $f_2^h$ of –5. So, after merging these derived fragments with the corresponding primary fragments, we get horizontal fragments $f_1^h$:{$I_3$}, $f_2^h$:{$I_1$, $I_2$, $I_3$, $I_4$, $I_5$, $I_6$} and $f_3^h$:{$I_2$, $I_6$}. Here, $I_2$, $I_3$ and $I_6$ are overlapping in different fragments. Applying affinity rule 2.2, $I_2$, $I_3$, and $I_6$ have maximum affinity of 60, 60 and 60 with $f_2^h$ respectively. Hence, the horizontal fragment is $f_1^h$:{$I_1$, $I_2$, $I_3$, $I_4$, $I_5$, $I_6$}.

For complex methods, we are assuming that instances $I_1$ and $I_3$ of class Company are invoked by methods of instances in fragment $f_1^h$ of class AutoCompany. Similarly, instances of $f_2^h$ of class AutoCompany invoke instance $I_2$ of class Company. Therefore, the derived fragments of class Company are $f_1^d$:{$I_1$, $I_3$} and $f_2^d$:{$I_2$}. We also assuming that, the above horizontal fragments of each class are the primary fragments. Applying affinity rule 2.1, $f_1^d$ has maximum affinity with $f_1^h$ of 0 and $f_2^d$ has maximum affinity with $f_1^h$ of –5. Hence, after merging $f_1^d$ and $f_2^d$ with $f_1^h$, the horizontal fragments of class Company is $f_1^h$:{$I_1$, $I_2$, $I_3$}. Now, the derived fragments of class Vehicle is $f_1^d$:{$I_1$, $I_2$, $I_3$, $I_4$, $I_5$, $I_6$} due to the complex attribute (Company). Applying affinity rule 2.1, $f_1^d$ has maximum affinity with $f_1^h$ of 40. Hence, after merging $f_1^d$ with $f_1^h$, the horizontal fragment of class Vehicle is $f_1^h$:{$I_1$, $I_2$, $I_3$, $I_4$, $I_5$, $I_6$}.

In [MK95], Savonnet and Yetongnon present a qualitative approach for class fragmentation in distributed object-oriented databases. They create a set of partition trees from class dependency graph (CDG). The CDG represents the inheritance, class composition and method nesting hierarchies. Each class of original object-oriented (OO) schema belongs to only one partition tree. Based on weighted factor of a node, this algorithm selects a root node for partition tree from the CDG. Then, it removes this node from the CDG as well as the descendant of that node and creates a partition tree. It repeats this process until CDG is empty. Then, this algorithm horizontally fragments classes of each partition by fragmenting the root class of the tree and propagating this fragmentation through other descendant classes of the tree. Hence,

primary fragmentation is defined on root class of the partition tree, but derived fragmentation is defined on other classes.

In [RZ95], Ravat and Zurfluh, present a fragmentation algorithm used for horizontal fragmentation with less time complexity than [OV99] and [CNP82]. They introduce predicate affinity, which describes the distance between two predicates. They group these predicates with higher affinity in a horizontal fragment. Each row in usage matrix represents a method used for horizontal fragmentation and each column represents a simple predicate. If the method "i" uses the simple predicate "j" then $C_{ij} = 1$ otherwise 0, where $C_{ij}$ represent the cell of $i^{th}$ row and $j^{th}$ column of the matrix. The predicate affinity matrix can be generated from the predicate usage matrix to represent the access frequency. They use the Bond Energy Algorithm to cluster and binary partitioning algorithm to partition the matrix. Every sub matrix contains a set of simple predicates, which must be linked using OR or AND to generate predicate term. Simple predicates with the same attribute are "ORed", and the predicates with different attributes are "ANDed". At last, they add a remaining fragment, which is the negation of disjunction of predicate terms. Predicate inclusions and predicate implications could be considered to reduce the number of predicates in every fragment.

In [BKS97], Bellatreche, Karlapalem, and Simonet, also introduce algorithm for both primary and derived horizontal fragmentation. The problem with the algorithm of [EB95] is that, the minterms generated by the algorithm is exponential with respect to the number of predicates. In [BKS97], by creating predicate usage and affinity matrix, author overcomes the problem. The predicate usage matrix of a class C contains queries as rows and predicates as columns, which is similar to the usage and affinity matrix of horizontal fragmentation algorithm described by Ravat and Zurfluh in [RZ95].

In [EZ99a], Ezeife and Zheng, present dynamic horizontal fragmentation of object-oriented database. The goal of this research is to analyze and determine the requirements for re-fragmentation. Then, it re-fragments the system from scratch by using algorithm described in [EB95]. To evaluate the performance, this algorithm uses two components for dynamic object horizontal distribution design. These are: (1) a mechanism for determining the performance threshold of an object horizontal distributed system, which is evaluated based on the partition evaluator measure and (2) a monitor algorithm for keeping track of changes in the system. The object horizontal partition evaluator algorithm immediately following a fragmentation and

allocation, computes the best system partition evaluation (PE) value, using application queries, access frequencies, class inheritance hierarchy and class composition hierarchy. This algorithm computes the system performance threshold using the computed best partition evaluator and system change variable (S). The system change variable measures, how frequently the system's inputs are changed. Monitoring algorithm keeps track of all changes in the input data and runs periodically. Then, it determines the current PE value using the object horizontal partition evaluator algorithm. If the current PE value is greater than the threshold value, it calls the system for re-fragmentation. The advantages of this system are:

- It eliminates the need for a full static requirements analysis and provides input for measuring system performance in order to determine when a re-fragmentation should be initiated.
- Dynamic fragmentation saves processing cost by eliminating the cost of full requirements study before every re-fragmentation.
- It improves overall system performance by detecting when the system performance drops below the threshold.

## 2.3 Performance Evaluation of Distributed Object-Oriented Database System

In case of distributed system, when data are updated due to addition and deletion the system performance may degrade. Performance evaluation for OODBMS is necessary for both designer and user. Designer needs to optimize the efficiency and adjust parameters of a system. The user needs to know the feasibility of the system for him.

In [CMVN93], Chakravarthy, Muthuraj, Varadarajan and Navathe, introduce an objective function - Partition Evaluator (PE), for vertical partitioning relations in distributed databases and its analysis, to compare and evaluate different algorithms that use the same input in distributed database design process. As the overall transaction processing cost in a distributed environment consists of local transaction processing cost and the remote transaction processing cost, they divide the partition evaluator into two parts – one is used to evaluate irrelevant local attribute access cost, the other is used to evaluate relevant remote attribute access cost. The irrelevant local attribute cost term measures the local processing cost of transactions due to irrelevant attributes of fragments by assuming all the data fragments required by a transaction are available

locally. The relevant remote attribute access term measures the remote processing cost, due to relevant attributes of data fragments that are accessed remotely by transactions. This partition evaluator has the flexibility to incorporate other information, such as type of queries (retrieval/updates), allocation information about the partitions, remote processing cost (transmission cost) and the transaction usage pattern at any particular site. The partition evaluator can be used as a basis for developing algorithms to create fragments of a relation.

In [EZ99b], Ezeife and Zheng, represent an algorithm (OHPE) to evaluate the performance of horizontal fragmentation of object-oriented database. The authors present a modified objective function, partition evaluator (PE), which is suitable for measuring the irrelevant local object instance cost ($E^2_M$) and relevant remote object instance cost ($E^2_R$) for horizontally fragmented object classes that are placed at distributed sites. The overall performance penalty for the fragments is the sum of these two performance penalties. For measuring the performance of a scheme, the object horizontal partitioning scheme computes horizontal fragments, which are allocated at distributed sites. In the second step, it transforms the available input data to obtain application object instance sets. The algorithm computes the application object instance set $[AI]_{qc}$ for every application $q$ at each site $c$. Then, application object instance sets and the fragments constitute input to the OHPE algorithm, which measures the PE value for the system. Each PE value corresponds to the total penalty cost incurred by each scheme through both local irrelevant access costs and remote relevant access costs. A lower PE value means less performance penalty and thus, a better performance.

## 2.4 OODBMS Application in Real World

This section shows some real world applications using OODBMS. Nowadays, object-oriented database is the major storage system for complex data structure.

In [GA98], Garland and Anthony from Motorola Inc., introduced the usefulness of Objectivity database into the IRIDIUM project. The IRIDIUM project involves a large number of components, including satellites in low earth orbit, and ground stations. System Control Segment (SCS), which is central to the entire IRIDIUM system, is developed using object-oriented (OO) software and development technique. Objectivity/DB has been successfully used in several SCS ground subsystems. The authors mentioned from their experience that objectivity/DB is a key element of the ground system software.

In [FEB98] Futtersack, Espert and Bolf from research and development (R & D) division of Electricite De France (EDF) represented a technique for managing documents including technical reports, project proposal and organization chart with Standard Generalized Markup Language (SGML) and object-oriented database. As they had 80,000 SGML documents and 60,000 Hyperindex documents, they used SGML for retrieval of documents from the $O_2$ object-based, which stores the documents as a tree structure. In that project, they used Search '97 full-text engine for searching text into documents. Authors concluded that they acheived better performance by integrating these three different technologies, namely an object database, a full-text retrieval engine and a web interface. The object-oriented database also demonstrated that it was very efficient and well suited for the persistence mechanism for a structured document management system.

In [HAG98], Hansen, Adams and Gracio, from Information Systems Department of Pacific Norhtwest National Laboratory (PNNL) submitted and presented a paper, which describes the experience that their group has had using ObjectStore as a database for a Scientific Data Management (SDM) System.

In [Th01], the author showed the data model for Geographical Information System (GIS) to capture real world data. Depending on the application, GIS has different name: (1) Geographic Database – it stores geographic data, (2) Land Information System (LIS) – it includes property and cadastral applications, (3) Automated Mapping/Facility Management (AM/FM) – it includes utilities management applications, and (4) Natural Resource Information Systems (NRIS) – it includes natural resource management applications. Data models for these different types of applications are different. Traditional relational databases are not designed for holding complex data models and large volume of data that are involved in building and ensuring the ongoing integrity of a real world geographic mapping database. Recently, object-oriented geospatial databases and associated mapping products have appeared which provide the technology to step into a new world of active objects and product independent geodata storage [Ha98].

Multimedia information systems (MMISs) is much more popular on the web. Both representation and retrieval of the complex and multifaceted multimedia data are not easily handled with the flat relational model and require new data models. The object-oriented data models provide powerful abstraction as well as structural and behavioral mechanisms to represent the complex multimedia information specifying the database schemas [RPMP99].

# Chapter 3: Incremental Object Horizontal Fragmentation

The design of distributed databases requires solutions to several interrelated problems: data fragmentation, allocation and local optimization [CMVN93]. A distributed object based systems (DOBS) requires class fragmentation and allocation, to get the maximum performance by transferring minimum data at different sites [EB95]. By reducing the amount of irrelevant data access and the amount of unnecessary data transfer, fragmentation improves the performance of an application [EB95]. Though fragmentation and allocation are the two different aspects of distribution design, distribution in the object world brings new complexities. All algorithms fragment the object based system from scratch. This chapter describes a new algorithm to re-fragment the object based system incrementally.

Section 3.1 defines several terms in order to present the algorithm formally. Section 3.2 proposes the new incremental object horizontal fragmentation (IOHF) algorithm for object-oriented database. Section 3.3 shows the correctness of IOHF with respect to OOHF algorithm.

## 3.1 Definitions

Before presenting algorithm for incremental object horizontal fragmentation, this section introduces some more definitions.

**Definition 3.1** *Maximum object-fragment affinity bond of a class, mofaffbond($I_i$, C)* is the highest affinity between any object of a class and any fragment of the class at time of initial fragmentation. For example, $I_1$, $I_2$ and $I_3$ are three instances of a class C. The fragments of the class are $f_1$:{$I_1$, $I_2$} and $f_2$:{$I_3$}. Assuming access frequencies are acco($I_1$) = 10, acco($I_2$) = 10 and acco($I_3$) = 5. The object fragment affinity, between object $I_1$ and fragment $f_1$, is ofaff($I_1$, $f_1$) = affo($I_1$, $I_2$) = acco($I_1$) + acco($I_2$) = 10 + 10 = 20. Similarly, other object fragment affinities of the class are ofaff($I_1$, $f_2$) = 15, ofaff($I_2$, $f_1$) = 20, ofaff($I_2$, $f_2$) = 15, ofaff($I_3$, $f_1$) = 30, and ofaff($I_3$, $f_2$) = 0. Here, object $I_3$ has the highest affinity of 30 with fragment $f_1$. Then, maximum object-fragment affinity bond of class C is 30 i.e., mofaffbond($I_3$, C) = 30.

**Definition 3.2** *Maximum fragment-fragment affinity bond of a class, mfaffbond($F_i$,$F_j$,C)* is the highest affinity between any two fragments of a class at time of initial fragmentation, where $i \neq j$.

For example, $f_1\{I_1, I_2\}$, $f_2\{I_3, I_4\}$ and $f_3\{I_5\}$ are fragments of a class C at time of initial fragmentation. Assuming, the access frequencies are $acco(I_1) = 10$, $acco(I_2) = 10$, $acco(I_3) = 5$, $acco(I_4) = 5$ and $acco(I_5) = 5$. Fragment-fragment affinity, between $f_1$ and $f_2$, is $aff(f_1, f_2)$ = Relevant access $(f_1, f_2)$ – Irrelevant access $(f_1, f_2) = 0 - acco(I_1) - acco(I_2) - acco(I_3) - acco(I_4) = -30$. Similarly, other fragment-fragment affinities of the class are $aff(f_1, f_3) = -25$ and $aff(f_2, f_3) = -15$. Here, fragment $f_2$ has maximum fragment-fragment affinity of –15 with fragment $f_3$. Then maximum fragment-fragment affinity bond of class C is –15 i.e., $mfaffbond(F_2,F_3,C) = -15$.

**Definition 3.3** *Derived access frequency dacco(I)* of an instance object (I) is the sum of the access frequencies of all the owner objects. Thus, $dacco(I) = \Sigma_i^k acco(I_i)$, where $I_i$ of owner class points to I of member class. For example, class B is a subclass of class A. Instance $I_5$ of class A is the pointer of instances $I_1$ and $I_2$ of class B. Then derived access frequency of $I_5$ is the sum of access frequency of $I_1$ and $I_2$ i.e., $dacco(I_5) = acco(I_1) + acco(I_2)$.

## 3.2 The Proposed Incremental Object Horizontal Fragmentation (IOHF) Algorithm

This section presents the proposed incremental object horizontal fragmentation algorithm for object-oriented database system, called IOHF. This algorithm fragments class objects horizontally and incrementally, by using changes in input data with the previous fragments to obtain new sets of best fragments. There are four types of changes, which are involved in degrading the performance of a distributed system. These are 1) changes in query access pattern, 2) changes in access frequencies, 3) changes in database schema (class inheritance and class composition hierarchy) and 4) changes in class instances. There are three basic types of changes: old one can be deleted, new one can be added or existing input can change its location. For example, new queries might start accessing a class, some of the old queries accessing a class may no longer be used or some queries might stop accessing one class but start accessing another class. Application queries, query access frequencies, instance objects, and object-oriented database schema (class inheritance and class composition hierarchy) are used as input to this algorithm.

This algorithm performs a sequence of actions on existing fragments to handle each of the four types of changes. For each of these changes of input data, the algorithm presents a

**Algorithm 3.1:** (Algorithm (IOHF – Algorithm for Performing Incremental Horizontal Fragmentation of Classes – Part I))

**Algorithm IOHF – Part I**

**Input:**
1. Previous Horizontal Fragments and their Minterm Predicates
2. Changes in Application Queries Access Pattern ($\Delta AF$)
3. Changes in Application Queries Access Frequencies ($\Delta AQ$)
4. Changes in class inheritance hierarchy ($\Delta CH$)
5. Changes in class composition hierarchy ($\Delta CCH$)
6. Changes in instance objects of classes (*ΔIS fragments)*
7. Existing fragmentation input data (AQ, AF, CH, CCH, IS)

**Output:** Updated Set of Horizontal Fragments of Classes ($F^H_N$)

begin

1. // To handle changes in Application Access Pattern, we do the following://

A. For all New Application Queries Arriving for a class do

begin

*1.a.1.* Obtain new primary horizontal fragments ($F^T$) of each fragments of the class using all of the instance objects of the fragments.

*1.a.2.* Obtain new derived-horizontal-fragment ($F^T_d$) of all member classes of this class using all of their instance-objects of each fragment.

*1.a.3* Merge the new derived fragments ($F^T_d$) with new primary fragments ($F^T$) using the affinity rules.

*1.a.4.* Merge the existing horizontal fragments, $F^H_k$ of those classes which have owner class, with new fragments $F^T_i$ it has the highest affinity aff($F^H_k$, $F^T_i$) with.

end // end of 1.A sequence//

B. For all Application Queries Deleted for a class from do

begin

*1.b.1.* Obtain primary horizontal fragments ($F^P$) of the class using all of the class's instance objects, applying the query.

*1.b.2.* Obtain derived horizontal fragments ($F^d$) of all member classes of this class using all of their instance objects.

*1.b.3.* By intersecting each fragments (both primary and derived) with existing horizontal fragments obtain the clustered fragment groups $f_j^{Ci} = f_i^T \cap f_j^{oh}$.

*1.b.4.* After subtracting acco(I) and dacco(I) from total access frequency of an object for deleting queries, merge $f_j^{Ci}$ with $f_k^{Cm}$ it has maximum affinity with, where $i \neq m$.

end // end of 1.B sequence //

C.For all Application Queries for a class moving from a site a to another site b do

begin

1.c.1. Execute sequence 1.B above to delete it.

1.c.2. Execute sequence 1.A above to add it.

end // End of 1.C sequence //

end // end of IOHF sequence 1 only//

**Figure 3.2.1: The Algorithm Incremental Horizontal Fragmentation (Part I)**

**Algorithm 3.2:** (Algorithm (IOHF – Algorithm for Performing Incremental Horizontal Fragmentation of Classes – Part II))

**Algorithm IOHF – Part II**

begin

2. // To handle change in Application Query Access frequencies, we do the following://

begin

*2.1.* Compute the affinity measure between every instance object I, in a class and all of the class's fragments, ofaff(I, $F^H_i$) using the change in application access frequencies. Then, re-assign I to the fragment, $F^H_k$ with highest affinity, if this highest affinity is larger than maximum affinity bond moaffbond(I, C) of the class. Maximum affinity bond of a class is the highest affinity between any object of a class and a fragment at time of initial fragmentation.

*2.2.* Obtain derived horizontal fragments ($F^d$) of all member classes of this class using all of their instance objects due to the new fragments.

*2.3.* Merge derived fragments $F^d$ with the existing fragments $F^H$ that has highest affinity.

end // end of sequence 2//

3. // To handle change in class inheritance and composition hierarchies, we do the following://

A. For all new class added to a link in the hierarchy, do

begin

*3.a.1.* Make all its new superclasses/containing classes (member classes) of this class and all its new subclasses/contained classes (owner classes) of this class.

*3.a.2.* For each new owner class of the class, obtain the derived horizontal fragment of the class, $F^d$.

*3.a.3.* Merge all new derived fragments $F^d$ with existing fragments $F^h$ that has highest affinity aff($F^d_i$, $F^h_k$) with.

*3.a.4.* For each new member class of this class, obtain the derived horizontal fragment of the class $F^d$.

*3.a.5.* Merge each derived fragment of the class, $F^d$, with the existing horizontal $F^H_k$ it has the highest affinity aff($F^d$, $F^H_k$) with.

end // end of 3.A sequence //

B. For a class deleted from a link in the hierarchy, do

begin

*3.b.1.* For each member class of this class, obtain the derived horizontal fragment based on the class, $F^d$.

*3.b.2.* After subtracting dacco(I) from total access frequency of each object for deleting class, merge each derived fragment, $F^d$, with its existing horizontal fragments $F^H_k$ it has the highest affinity aff($F^d$, $F^H_k$) with.

end // End of 3.B sequence //

C. For a class moving from one part of the link to another in the hierarchy, do

begin

*3.c.1.* Execute sequence 3.B above to delete it.

*3.c.2.* Execute sequence 3.A above to add it.

end // End of 3.C sequence //

end // end of IOHF sequence 2 and 3 only//

**Figure 3.2.2: The Algorithm Incremental Horizontal Fragmentation (Part II)**

**Algorithm 3.3:** (Algorithm (IOHF – Algorithm for Performing Incremental Horizontal Fragmentation of Classes – Part III))

**Algorithm IOHF – Part III**
begin
4. // To handle change in Instance objects, we do the following://
A. For new instance objects arriving, do
begin
*4.a.1.* Apply all minterms to the new instances of a class to get primary fragments.
*4.a.2.* For each primary fragment of a class, apply the same minterm that generates it, to all of the existing fragments of the class. This process finds only one fragment for each minterm from existing fragments or none. If the process finds a fragment, then merge the primary fragment with that existing fragment. Otherwise, the primary fragment will be a new primary fragment for the class.
*4.a.3.* Apply the merging rule to merge all derived fragments of this class with any old or new primary fragment of the class it has highest affinity with.
*4.a.4.* Propagate the presence of these new instance objects of this class by using every fragment a new instance object is in to define a derived fragment of the member class.
*4.a.5.* Merge each derived fragments of the member class, $F^h_d$, with its old horizontal fragment $F^H_k$ it has the highest affinity aff($F^h_d$, $F^H_k$) with.
end // end of 4.A sequence //
B. For instance objects deleted from a class, do
begin
*4.b.1.* Remove them from fragments of the class and generates new fragment $f^N$.
*4.b.2.* Obtain derived horizontal fragments ($F^d$) of all member classes of this class using all of their instance objects.
*4.b.3.* By intersecting each derived fragments with new fragments, obtain the clustered-fragment $f^{C_i} = f_i^T \cap f_j^N$.
*4.b.4.* After subtracting dacco(I) from total access frequency of each deleting object for deleting instance, merge $f_j^{Ci}$ with $f_j^N$ it has maximum affinity with.
end // End of 4.B sequence //
C. For instance objects moving from a class to another, do
begin
*4.c.1.* Execute sequence 4.B above to delete it
*4.c.2.* Execute sequence 4.A above to add it
end // End of 4.C sequence //
end // end of IOHF sequence 4 only //

**Figure 3.2.3: The Algorithm Incremental Horizontal Fragmentation (Part III)**

sequence of steps to handle (a) adding the necessary input to the system, (b) deleting the necessary input from the system and (c) moving the necessary input from one location to another. The step c is generally treated as executing step b followed by an execution of step a. The detailed algorithm is presented in figure 3.2.1 (part I), figure 3.2.2 (part II) and figure 3.3.3 (part III). The following sections describe the sequence of actions in this algorithm for each of the four types of changes.

**Country {CnID, CnName, CnPop, GetName, GetPop}**
$I_1$ {Cn01, Canada, 4000000}
$I_2$ {Cn02, USA, 8000000}

**State {CnID, StID, StName, StPop, GetName, GetPop}**
$I_1$ {Country Pointer1, St01, Ontario, 1500000}
$I_2$ {Country Pointer1, St02, Quebec, 1000000}
$I_3$ {Country Pointer2, St03, New York, 2000000}

**City {StID, CiID, CiName, CiPop GetStID, GetName, GetPop}**
$I_1$ {State Pointer1, Ci01, Toronto, 80000}
$I_2$ {State Pointer1, Ci02, Windsor, 40000}
$I_3$ {State Pointer2, Ci03, Montreal, 50000}
$I_4$ {State Pointer3, Ci04, New York, 90000}
$I_5$ {State Pointer3, Ci05, Queens, 60000}

**School {CiID, ScID, ScName, ScStudent, ScCategory, GetName, GetStudent, GetCategory}**
$I_1$ {City Pointer1, Sc01, Uof Toronto, 4500, University}
$I_2$ {City Pointer1, Sc02, Y University, 4000, University}
$I_3$ {City Pointer2, Sc03, U of Windsor, 3000, University}
$I_4$ {City Pointer2, Sc04, ABC School, 500, Elementary}
$I_5$ {City Pointer4, Sc05, NY University, 3500, University}
$I_6$ {City Pointer5, Sc06, M School, 400, Elementary}

**Hospital {CiID, HoID, HoName, HoBeds GetName, GetBeds}**
$I_1$ {City Pointer1, Ho01, Toronto Hospital, 200}
$I_2$ {City Pointer2, Ho02, Windsor Hospital, 300}
$I_3$ {City Pointer2, Ho03, WS Hospital, 150}
$I_4$ {City Pointer3, Ho04, Montreal Hospital, 100}
$I_5$ {City Pointer4, Ho05, NY Hospital, 300}
$I_6$ {City Pointer5, Ho06, Queens Hospital, 150}

**Road {StID, RoID, RoName, RoUOM GetName, GetUOM}**
$I_1$ {State Pointer1, Ro01, 401, km}
$I_2$ {State Pointer1, Ro02, 403, km}
$I_3$ {State Pointer1, Ro03, EC Express, km}
$I_4$ {State Pointer2, Ro04, 401, km}
$I_5$ {State Pointer3, Ro05, I75, mile}
$I_6$ {State Pointer3, Ro06, I94, mile}

**Highway {RoID, HiID, HiSpeed, HiLanes GetSpeed, GetLanes}**
$I_1$ {Road Pointer1, Hi01, 80, 4}
$I_2$ {Road Pointer1, Hi02, 100, 4}
$I_3$ {Road Pointer1, Hi03, 100, 8}
$I_4$ {Road Pointer2, Hi04, 100, 4}
$I_5$ {Road Pointer3, Hi05, 100, 4}
$I_6$ {Road Pointer4, Hi06, 100, 6}
$I_7$ {Road Pointer5, Hi07, 75, 4}
$I_8$ {Road Pointer6, Hi08, 65, 6}

**UrbanRoad {RoID, UrID, UrSpeed, UrLanes GetSpeed, GetLanes}**
$I_1$ {Road Pointer1, Ur01, 50, 4}
$I_2$ {Road Pointer1, Ur02, 60, 4}
$I_3$ {Road Pointer2, Ur03, 50, 4}
$I_4$ {Road Pointer3, Ur04, 60, 6}
$I_5$ {Road Pointer5, Ur05, 35, 3}
$I_6$ {Road Pointer6, Ur06, 40, 4}

**Figure 3.2.4: Sample Object Database Schema**

To facilitate the description of incremental object horizontal fragmentation algorithm, we use classes Country, State, City, Road, School, Hospital, Urbanroad and Highway as a sample object database as in figure 3.2.4. Each class has its own attributes, methods and instances. For example, ID (CnID), Name (CnName) and Population (CnPop) are attributes of class Country. GetName and GetPop are methods of class Country. $I_1$ and $I_2$ are instances of class Country. Again, each instance of a subclass contains pointer to instance of its super class. For example, $I_2$ of class City contains pointer to instance $I_1$ of its super class State. Figure 3.2.5 shows the class inheritance hierarchy, where class State is a subclass of Country, class City and class Road are subclasses of class State, class Hospital and class School are subclasses of City and class

Urabanroad and class Highway are subclasses of class Road. According to OOHF algorithm, the class composition hierarchy the object database is shown in figure 3.2.6.

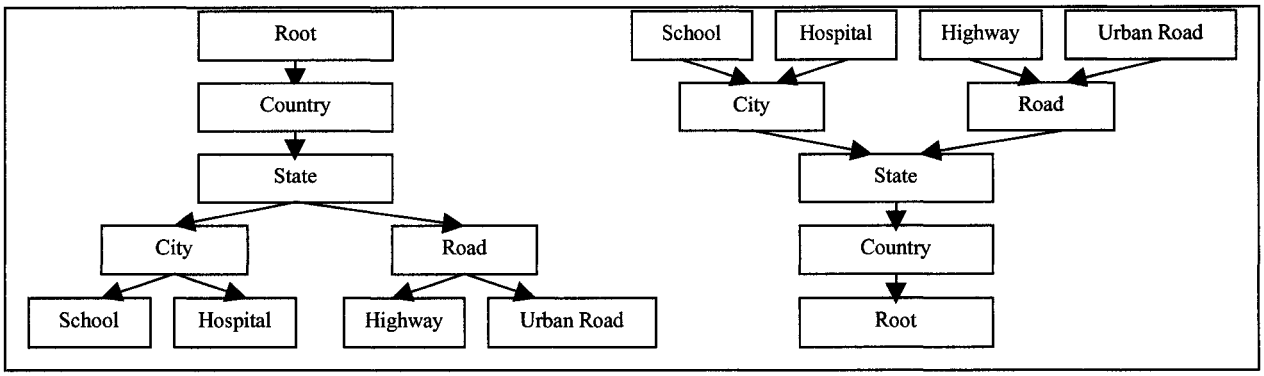


**Figure 3.2.5:Class Inheritance Hierarchy** **Figure 3.2.6: Class Composition Hierarchy**

**Query 1:** This query groups objects of class *Hospital* according to their accommodation. Query requires method GetBeds of class *Hospital* and predicates from the query are $P_1$: HoBeds > 150 and $P_2$: HoBeds ≤ 150.

**Query 2:** This query groups objects of class *School* according to their category. Query requires method GetCategory of class *School* and predicates from the query are $P_1$: ScCategory = "Univeristy" and $P_2$: ScCategory = "Elementary".

**Query 3:** This query groups objects of class *City* according to their population. Query requires method GetPop of class *City* and predicates from the query are $P_1$: CiPop > 60,000 and $P_2$:CiPop ≤ 60,000.

**Query 4:** This query groups objects of class *State* according to their name. Query requires method GetName of class *State* and predicates from the query are $P_1$: StName = "Ontario", $P_2$: StName = "Quebec" and $P_3$: StName = "New York".

**Query 5:** This query groups objects of class *Highway* according to their speed limit. Query requires method GetSpeed of class Highway and predicates from the query are $P_1$: HiSpeed ≥ 100 km or 75 miles and $P_2$: HiSpeed < 100 km or 75 miles.

**Figure 3.2.7: Application Queries**

| | |
|---|---|
| Hospital: | $acco(I_1) = 20$, $acco(I_2) = 20$, $acco(I_3) = 10$, $acco(I_4) = 10$, $acco(I_5) = 20$, $acco(I_6) = 10$ |
| School: | $acco(I_1) = 5$, $acco(I_2) = 5$, $acco(I_3) = 5$, $acco(I_4) = 10$, $acco(I_5) = 5$, $acco(I_6) = 10$ |
| Highway: | $acco(I_1) = 5$, $acco(I_2) = 10$, $acco(I_3) = 10$, $acco(I_4) = 10$, $acco(I_5) = 10$, $acco(I_6) = 10$, $acco(I_7) = 10$, $acco(I_8) = 5$ |
| City: | $acco(I_1) = 10$, $acco(I_2) = 5$, $acco(I_3) = 5$, $acco(I_4) = 10$, $acco(I_5) = 5$ |
| Road: | $acco(I_1) = 5$, $acco(I_2) = 10$, $acco(I_3) = 10$, $acco(I_4) = 10$, $acco(I_5) = 10$, $acco(I_6) = 5$ |
| State: | $acco(I_1) = 5$, $acco(I_2) = 10$, $acco(I_3) = 5$ |
| Country: | $acco(I_1) = 10$, $acco(I_2) = 5$ |

**Figure 3.2.8: Access Frequencies**

Figure 3.2.7 and figure 3.2.8 show the application queries and access frequencies of classes respectively. Based on these application queries and access frequencies, OOHF algorithm generates horizontal fragments for each class as in figure 3.2.9. For example, class School has two horizontal fragments, which are $f_1^h$:{$I_1$, $I_2$, $I_3$, $I_5$} and $f_2^h$:{$I_4$, $I_6$}. So, the class inheritance hierarchy, composition hierarchy, application queries, access frequencies, database schema and horizontal fragments are used as input to the IOHF algorithm.

| | **Site 1** | **Site 2** |
|---|---|---|
| **School** | $f_1^h$: {$I_1$, $I_2$, $I_3$, $I_5$} | $f_2^h$: {$I_4$, $I_6$} |
| **Hospital** | $f_1^h$: {$I_1$, $I_2$, $I_5$} | $f_2^h$: {$I_3$, $I_4$, $I_6$} |
| **Highway** | $f_1^h$: {$I_2$, $I_3$, $I_4$, $I_5$, $I_6$, $I_7$} | $f_2^h$: {$I_1$, $I_8$} |
| **City** | $f_1^h$: {$I_1$, $I_2$, $I_4$} | $f_2^h$: {$I_3$, $I_5$} |
| **Road** | $f_1^h$: {$I_1$, $I_2$, $I_3$, $I_4$, $I_5$} | $f_2^h$: {$I_6$} |
| **State** | $f_1^h$: {$I_1$, $I_2$, $I_3$} | |
| **Country** | $f_1^h$: {$I_1$, $I_2$} | |

**Figure 3.2.9: Horizontal Fragments**

### *3.2.1 Changes in Application Access Pattern*

When new queries are added to the distributed system, IOHF algorithm generates minterm predicates and applies these minterm predicates to each existing fragments to create new fragments. For example, based on new queries (figure 3.2.1.1), it generates minterm predicates for class Road, class School and class City.

**Query 1:** This query groups objects of class *Road* according to their unit of measurement. Query requires method GetUOM of class *Road* and predicates from the query are $P_1$: RoUOM = "km" and $P_2$: RoUOM = "mile".

**Query 2:** This query groups objects of class *School* according to their number of student. Query requires method GetStudent of class *School* and predicates from the query are $P_1$: ScStudent > 3000 and $P_2$: ScStudent ≤ 3000.

**Query 3:** This query groups objects of class *City* according to their State ID. Qurey requires method GetStID of class *City* and predicates from the query are $P_1$: StID = "State Pointer 1", $P_2$: StID = "State Pointer 2" and $P_3$: StID = "State Pointer 3".

**Figure 3.2.1.1: New Queries – Changes in Application Access Pattern**

The minterm predicates are:

Class School: $MM_1$: ScStudent > 3000

$MM_2$: ScStudent ≤ 3000

Class Road: $MM_1$: RoUOM = "KM" ∧ RoUOM ≠ "MILE"

$MM_2$: RoUOM ≠ "KM" ∧ RoUOM = "MILE"

Class City: $MM_1$: StID="State Pointer1" ∧ StID≠"State Pointer2" ∧ StID≠"State Pointer3"

$MM_1$: StID≠"State Pointer1" ∧ StID="State Pointer2" ∧ StID≠"State Pointer3"

$MM_1$: StID≠"State Pointer1" ∧ StID≠"State Pointer2" ∧ StID="State Pointer3"

Class school has two existing fragments, $f_1^{oh}$:{$I_1$, $I_2$, $I_3$, $I_5$} and $f_2^{oh}$:{$I_4$, $I_6$}. Applying $MM_1$ to fragments $f_1^{oh}$, we get $f_1^{np}$:{$I_1$, $I_2$, $I_5$}. Applying $MM_1$ to fragments $f_2^{oh}$, we get $f_2^{np}$:{$I_3$}. Similarly, applying $MM_2$ to fragments $f_1^{oh}$ and $f_2^{oh}$, we get $f_3^{np}$:{} and $f_4^{np}$:{$I_4$, $I_6$}. After eliminating fragments with empty sets (e.g., $f_3^{np}$), we get the new fragments of class School as $f_1^{np}$:{$I_1$, $I_2$, $I_5$}, $f_2^{np}$:{$I_3$} and $f_3^{np}$:{$I_4$, $I_6$}. Similarly, after applying $MM_1$ and $MM_2$ of class Road to fragments $f_1^{oh}$ and $f_2^{oh}$, we get $f_1^{np}$:{$I_1$, $I_2$, $I_3$, $I_4$}, $f_2^{np}$:{$I_5$} and $f_3^{np}$:{$I_6$}. Again, applying $MM_1$,

$MM_2$ and $MM_3$ of class City to fragments $f_1^{oh}$ and $f_2^{oh}$, we get $f_1^{np}$:{$I_1$, $I_2$}, $f_2^{np}$:{$I_4$}, $f_3^{np}$:{$I_3$} and $f_4^{np}$:{$I_5$}.

Now, based on these new fragments, it generates new derived fragments of all member classes of the class, using all of their instance objects. For example, the derived fragments of class City, based on new fragments $f_1^{np}$ of class School is $f_1^{nd}$:{$I_1$, $I_4$}. Similarly, the derived fragments of class City based on fragments of $f_2^{np}$ and $f_3^{np}$ of class School are $f_2^{nd}$:{$I_2$} and $f_3^{nd}$:{$I_2$, $I_5$}. Derived fragments of class State, based on primary fragments of class City are $f_1^{nd}$:{$I_1$}, $f_2^{nd}$:{$I_3$}, $f_3^{nd}$:{$I_2$} and $f_4^{nd}$:{$I_3$}, based on derived fragments of class City are $f_5^{nd}$:{$I_1$, $I_3$}, $f_6^{nd}$:{$I_1$} and $f_7^{nd}$:{$I_1$, $I_3$} and based on primary fragments of class Road are $f_8^{nd}$:{$I_1$, $I_2$}, $f_9^{nd}$:{$I_3$} and $f_{10}^{nd}$:{$I_3$}. Similarly, derived fragments of class Country, based on derived fragments of class State are $f_1^{nd}$:{$I_1$, $I_2$}, $f_2^{nd}$:{$I_1$}, $f_3^{nd}$:{$I_1$, $I_2$}, $f_4^{nd}$:{$I_1$}, $f_5^{nd}$:{$I_2$}, $f_6^{nd}$:{$I_2$}, $f_7^{nd}$:{$I_1$}, $f_8^{nd}$:{$I_2$}, $f_9^{nd}$:{$I_1$} and $f_{10}^{nd}$:{$I_2$}.

Now, it integrates these new derived fragments with new primary fragments by applying affinity rule 2.1 and affinity rule 2.2. For example, fragments $f_1^{nd}$, $f_2^{nd}$, and $f_3^{nd}$ of class City have maximum affinity of 0, -5 and 0 with $f_2^{np}$, $f_1^{np}$ and $f_4^{np}$ respectively. After merging these derived fragments with corresponding primary fragments, we get $f_1^{h}$:{$I_1$, $I_2$}, $f_2^{h}$:{$I_1$, $I_4$}, $f_3^{h}$:{$I_3$} and $f_4^{h}$:{$I_2$, $I_5$}. Here, $I_1$ and $I_2$ are overlapping in different fragments. Applying affinity rule 2.2, $I_1$ and $I_2$ are placed in $f_2^{h}$ and $f_4^{h}$ with maximum affinity of 20 and 10 respectively. Hence, the fragments are $f_1^{nh}$:{$I_1$, $I_4$}, $f_2^{nh}$:{$I_3$} and $f_3^{nh}$:{$I_2$, $I_5$}. Instances $I_1$, $I_2$ and $I_3$ of class State are overlapping in different fragments. Applying affinity rule 2.2, $I_1$, $I_2$ and $I_3$ are placed in $f_8^{nd}$, $f_8^{nd}$ and $f_5^{nd}$ with maximum affinity of 15, 15 and 10 respectively. Hence, the fragments are $f_1^{nh}$:{$I_1$, $I_2$} and $f_2^{nh}$:{$I_3$}. Applying affinity rule 2.2, instances $I_1$ and $I_2$ of class Country are placed in $f_1^{nd}$ with maximum affinity of 15 and 15 respectively. Hence, the fragment is $f_1^{nh}$:{$I_1$, $I_2$}.

Now, classes have their own existing horizontal fragments $f^{oh}$ (figure 3.2.8) and sets of new fragments $f^{np}$. It merges the existing fragments with new fragments based on affinity rule 2.1 and affinity rule 2.2, for those classes, which are member classes in composition hierarchy. For example, class School is not a member class of other classes, so the new fragments are the incremental horizontal fragments of the class. On the other hand, class City is a member class, so this algorithm merges the existing fragments with the new fragments. Fragments $f_1^{oh}$ and $f_2^{oh}$ of class City have maximum affinity of 15 and 0 with $f_1^{nh}$ and $f_2^{nh}$ respectively. After merging these existing fragments with new fragments, we get $f_1^{h}$: {$I_1$, $I_2$, $I_4$}, $f_2^{h}$ {$I_3$, $I_5$} and $f_3^{h}$ {$I_2$, $I_5$}. Here, $I_2$

and $I_5$ are overlapping in different fragments. Applying affinity rule 2.2, $I_2$ and $I_5$ are placed in $f_1^h$ and $f_2^h$ with maximum affinity of 30 and 10 respectively. Hence, the incremental horizontal fragments are $f_1^h$: $\{I_1, I_2, I_4\}$ and $f_2^h$: $\{I_3, I_5\}$. Fragments $f_1^{oh}$ and $f_2^{oh}$ of class Road have maximum affinity of 30 and 5 with $f_1^{nh}$ and $f_3^{nh}$ respectively. After merging these existing fragments with corresponding new fragments, we get $f_1^h$:$\{I_1, I_2, I_3, I_4, I_5\}$, $f_2^h$:$\{I_5\}$ and $f_3^h$:$\{I_6\}$. Here, $I_5$ is overlapping in different fragments. Applying affinity rule 2.2, $I_5$ is placed in $f_1^h$ with maximum affinity of 55. Hence, the final incremental fragments are $f_1^h$: $\{I_1, I_2, I_3, I_4, I_5\}$ and $f_2^h$:$\{I_6\}$. Fragment $f_1^{oh}$ of class State has maximum affinity of 10 with $f_1^{nh}$. After merging $f_1^{oh}$ with $f_1^{nh}$, we get $f_1^h$:$\{I_1, I_2, I_3\}$ and $f_2^h$:$\{I_3\}$. Here, $I_3$ is overlapping in both fragments. Applying affinity rule 2.2, $I_3$ is placed in $f_1^h$ with maximum affinity of 25. Hence, the incremental horizontal fragment is $f_1^h$: $\{I_1, I_2, I_3\}$. Fragment $f_1^{oh}$ of class Country has maximum affinity of 15 with $f_1^{nh}$. After merging $f_1^{oh}$ with $f_1^{nh}$, the incremental horizontal fragment is $f_1^h$: $\{I_1, I_2\}$.

The following table shows incremental horizontal fragments of classes, after adding queries.

| **Classes** | **Fragments using IOHF Algorithm** | | |
|---|---|---|---|
| *School* | $f_1^h$:$\{I_1,I_2,I_5\}$ | $f_2^h$:$\{I_3\}$ | $f_3^h$:$\{I_4,I_6\}$ |
| *City* | $f_1^h$:$\{I_1,I_2,I_4\}$ | $f_2^h$:$\{I_3,I_5\}$ | |
| *Road* | $f_1^h$:$\{I_1,I_2,I_3,I_4,I_5\}$ | $f_2^h$:$\{I_6\}$ | |
| *State* | $f_1^h$: $\{I_1,I_2,I_3\}$ | | |
| *Country* | $f_1^h$:$\{I_1,I_2\}$ | | |

**Table 3.2.1.1: Fragments of classes – Adding Queries**

When application access pattern changes due to deletion of queries, algorithm IOHF generates fragments of classes by applying minterm predicates to all of the instances of the class. For example, when we delete query 2 and query 3 of figure 3.2.7, it applies the minterm predicates for these queries to class School and class City respectively. The fragments generated by the minterm predicates of class School are $f_1^p$: $\{I_1, I_2, I_3, I_5\}$ and $f_2^p$: $\{I_4, I_6\}$. Similarly, the fragments for class City are $f_1^p$: $\{I_1, I_4\}$ and $f_2^p$: $\{I_2, I_3, I_5\}$.

Now, based on these fragments, it creates derived fragments of all member classes. For example, the derived fragments of class City, based on fragment $f_1^p$ and $f_2^p$ of class School are $f_1^d$:$\{I_1, I_2, I_4\}$ and $f_2^d$: $\{I_2, I_5\}$ respectively. Similarly, the derived fragments of class State, based

on fragments of class City are $f_1^d$:{$I_1$, $I_3$}, and $f_2^d$:{$I_1$, $I_2$, $I_3$} and based on the derived fragments of class City are $f_3^d$: {$I_1$, $I_3$} and $f_4^d$: {$I_1$, $I_3$}. The derived fragments of class Country, based on derived fragments of class State are $f_1^d$:{$I_1$, $I_2$}, $f_2^d$:{$I_1$, $I_2$}, $f_3^d$: {$I_1$, $I_2$} and $f_4^d$: {$I_1$, $I_2$}.

To remove the effect of the deleted queries from the existing fragments, it generates clustered fragments for each fragment. By intersecting each fragment (both primary and derived) with existing fragments of the class, it creates clustered fragments $f_j^{C_i}$. For example, after intersecting $f_1^p$:{$I_1$, $I_2$, $I_3$, $I_5$} and $f_1^{oh}$:{$I_1$, $I_2$, $I_3$, $I_5$} of class School, this algorithm generates $f_1^{I_1}$:{$I_1$, $I_2$, $I_3$, $I_5$}. Again, after intersecting $f_1^p$ with $f_2^{oh}$ of class School, it generates $f_2^{I_1}$:{}. Similarly, $f_2^p$ after intersecting with $f_1^{oh}$ and $f_2^{oh}$ generates $f_1^{I_2}$:{} and $f_2^{I_2}$:{$I_4$, $I_6$} respectively.

| **Class Name** | **Instance** | **acco(I)** | **dacco(I)** | | **Total** |
|---|---|---|---|---|---|
| **School** | $I_1$ | 5 | 0 | | 5 |
| | $I_2$ | 5 | 0 | | 5 |
| | $I_3$ | 5 | 0 | | 5 |
| | $I_4$ | 10 | 0 | | 10 |
| | $I_5$ | 5 | 0 | | 5 |
| | $I_6$ | 10 | 0 | | 10 |
| **Class Name** | **Instance** | **acco(I)** | **dacco(I) (Based on Class School)** | **dacco(I) (Based on Class Hospital)** | **Total** |
| **City** | $I_1$ | 10 | $5(I_1) + 5(I_2)$ | $20(I_1)$ | 40 |
| | $I_2$ | 5 | $5(I_3) + 10(I_4)$ | $20(I_2) + 10(I_3)$ | 50 |
| | $I_3$ | 5 | 0 | $10(I_4)$ | 15 |
| | $I_4$ | 10 | $5(I_5)$ | $20(I_5)$ | 35 |
| | $I_5$ | 5 | $10(I_6)$ | $10(I_6)$ | 25 |
| **Class Name** | **Instance** | **acco(I)** | **dacco(I) (Based on Class City)** | **dacco(I) (Based on Class Road)** | **Total** |
| State | $I_1$ | 5 | $10(I_1) + 5(I_2)$ | $5(I_1)+10(I_2)+10\ (I_3)$ | 45 |
| | $I_2$ | 10 | $5(I_3)$ | $10(I_4)$ | 25 |
| | $I_3$ | 5 | $10(I_4) + 5(I_5)$ | $10(I_5) + 5(I_6)$ | 35 |
| **Class Name** | **Instance** | **acco(I)** | **dacco(I) (Based on Class State)** | | **Total** |
| Country | $I_1$ | 10 | $5(I_1) + 10\ (I_2)$ | | 25 |
| | $I_2$ | 5 | $5(I_3)$ | | 10 |

**Note:** Within bracket of column dacco shows the instance of owner class. For example, 2nd row and 4th columns of class City represent the access frequency 5 and 10 of instance $I_3$ and $I_4$ of owner class School respectively.

**Figure 3.2.1.2: Primary and Derived Access Frequencies – Changes in Application Access Pattern**

For class City, $f_1^p$ generates $f_1^{I_1}$:{$I_1$, $I_4$} and $f_2^{I_1}$:{} after intersecting with $f_1^{oh}$ and $f_2^{oh}$ respectively. Similarly, $f_2^p$ generates $f_1^{I_2}$:{$I_2$} & $f_2^{I_2}${$I_3$, $I_5$}, $f_1^d$ generates $f_1^{I_3}$:{$I_1$, $I_2$, $I_4$} & $f_2^{I_3}$:{} and $f_2^d$ generates $f_1^{I_4}${$I_2$} & $f_2^{I_4}${$I_5$} for class City. For class State, $f_1^d$ generates $f_1^{I_1}$:{$I_1$, $I_3$}, $f_2^d$ generates $f_1^{I_2}$:{$I_1$, $I_2$, $I_3$}, $f_3^d$ generates $f_1^{I_3}$:{$I_1$, $I_3$} and $f_4^d$ generates $f_1^{I_4}$:{$I_1$, $I_3$}. For class Country, $f_1^d$ generates $f_1^{I_1}$:{$I_1$, $I_2$}, $f_2^d$ generates $f_1^{I_2}$:{$I_1$, $I_2$}, $f_3^d$ generates $f_1^{I_3}$:{$I_1$, $I_2$} and $f_4^d$ generates $f_1^{I_4}$:{$I_1$, $I_2$}.

Now, it subtracts acco(I) and dacco(I) from total access frequencies for each instance of the class and merges these fragments $f_j^{C_i}$ and $f_j^{C_m}$ by applying affinity rule 2.1 and affinity rule 2.2, where $i \neq m$. Figure 3.2.1.2 shows the access frequencies and derived access frequencies based on sample data and access frequencies of figure 3.2.4 and figure 3.2.8 respectively. Here, column "acco(I)" and "dacco(I)" represent the access frequencies and derived access frequencies of instances respectively. The column "Total" represents the sum of access frequencies and derived access frequencies (acco(I) + dacco(I)) for each instance. As class Student does not have any owner class, so the dacco(I) for each instances are 0. On the other hand, class Student and class Hospital are two owner classes of class City. Instance $I_1$ of class City is the superclass pointer of instances $I_1$ and $I_2$ of class School, so the column "dacco(I) (based on class School)" is the sum of access frequencies 5 and 5 of $I_1$ and $I_2$ of class School respectively. Similarly, instance $I_1$ of class Hospital points to the instance $I_1$ of class City, so the column "dacco(I) (based on class Hospital)" is the access frequencies of $I_1$ of class Hospital. Now, after subtracting the access frequencies of instances for query 2 from total access frequencies, the access frequencies for instances of class School are 0. Fragment $f_1^{I_1}$ of class School has maximum affinity of 0 with $f_2^{I_2}$, hence after merging $f_1^{I_1}$ with $f_2^{I_2}$, incremental horizontal fragment is $f_1^h$:{$I_1$, $I_2$, $I_3$, $I_4$, $I_5$, $I_6$}. For class City, after subtracting the access frequencies for query 3 and derived access frequencies for class School from total access frequencies, we get access frequencies of instances $I_1$, $I_2$, $I_3$, $I_4$ and $I_5$ as 20, 30, 10, 20 and 10 respectively. Now, $f_1^{I_1}$, $f_1^{I_2}$, $f_2^{I_2}$, $f_1^{I_3}$, $f_1^{I_4}$ and $f_2^{I_4}$ have maximum affinity of 10, 30, 0, 10, 30 and 0 with $f_1^{I_3}$, $f_1^{I_4}$, $f_2^{I_4}$, $f_1^{I_1}$, $f_1^{I_2}$ and $f_2^{I_2}$ respectively. After merging these fragments, we get $f_1^h$:{$I_1$, $I_2$, $I_4$}, $f_2^h$:{$I_2$}, $f_3^h$:{$I_3$, $I_5$}, $f_4^h$:{$I_1$, $I_2$, $I_4$}, $f_5^h$:{$I_2$} and $f_6^h$:{$I_3$, $I_5$}. For simplicity, keeping one of the identical fragments, we get $f_1^h$:{$I_1$, $I_2$, $I_4$}, $f_2^h$:{$I_2$} and $f_3^h$:{$I_3$, $I_5$}. Here, $I_2$ is overlapping in different fragments. Applying affinity rule 2.2, $I_2$ is placed in $f_1^h$ with maximum affinity of 100. Hence, the incremental horizontal fragments are $f_1^h$:{$I_1$, $I_2$, $I_4$} and $f_2^h$:{$I_3$, $I_5$}. For class State, after subtracting the derived access frequencies for

class City from total access frequencies, we get access frequencies of instances $I_1$, $I_2$, and $I_3$ are 30, 20 and 20 respectively. Now, $f_1^{I_1}$, $f_1^{I_2}$, $f_1^{I_3}$, and $f_1^{I_4}$ have maximum affinity of 50, 30, 50, and 50 with $f_1^{I_3}$, $f_1^{I_1}$, $f_1^{I_1}$, and $f_1^{I_1}$ respectively. After merging these fragments and keeping one of the identical fragments, we get $f_1^h$:{$I_1$, $I_3$} and $f_2^h$:{$I_1$, $I_2$, $I_3$}. Here, $I_1$ and $I_3$ are overlapping in different fragments. Applying affinity rule 2.2, $I_1$ and $I_3$ are placed in $f_2^h$ with maximum affinity of 100 and 90 respectively. Hence, the incremental horizontal fragment is $f_1^h$:{$I_1$, $I_2$, $I_3$}. For class Country, after subtracting the derived access frequencies for class State, we get access frequencies of instances $I_1$, and $I_2$ are 10 and 5 respectively. Now, $f_1^{I_1}$, $f_1^{I_2}$, $f_1^{I_3}$, and $f_1^{I_4}$ have maximum affinity of 15, 15, 15, and 15 with each other. After merging these fragments and keeping one of the identical fragments, we get the incremental horizontal fragment is $f_1^h$:{$I_1$, $I_2$}.

The following table shows incremental horizontal fragments of classes, after deleting queries.

| **Classes** | **Fragments using IOHF Algorithm** | |
|---|---|---|
| *School* | $f_1^h$:{$I_1$,$I_2$,$I_3$,$I_4$,$I_5$,$I_6$} | |
| *City* | $f_1^h$:{$I_1$,$I_2$,$I_4$} | $f_2^h$:{$I_3$,$I_5$} |
| *State* | $f_1^h$: {$I_1$,$I_2$,$I_3$} | |
| *Country* | $f_1^h$:{$I_1$,$I_2$} | |

**Table 3.2.1.2: Fragments of classes – Deleting Queries**

### *3.2.2 Changes in Application Query Access Frequencies*

IOHF algorithm computes affinity measure, ofaff(I, $F_i^H$), between every instance I in a class and all of the class's fragments, using new application access frequencies. If the highest affinity is greater than maximum affinity bond moaffbond(I, C) (definition 3.1) of the class, then it reassigns I to fragment $F_k^H$ to generate primary fragments. For example, assuming the new access frequencies of instances of class City are acco($I_1$) = 5, acco($I_2$) = 15, acco($I_3$) = 15, acco($I_4$) = 5 and acco($I_5$) = 15. Based on existing access frequencies (figure 3.2.8), $I_1$, $I_2$, $I_3$, $I_4$ and $I_5$ have maximum affinity of 35, 30, 40, 35 and 40 with existing fragments $f_1^{oh}$, $f_1^{oh}$, $f_1^{oh}$, $f_1^{oh}$ and $f_1^{oh}$ respectively. Hence, the maximum affinity bond at time of initial fragmentation is 40. With new access frequencies, $I_1$, $I_2$, $I_3$, $I_4$ and $I_5$ have maximum affinity of 40, 60, 30, 40 and 30 with $f_2^{oh}$ respectively. The existing fragments of class City are $f_1^{oh}$: {$I_1$, $I_2$, $I_4$} and $f_2^{oh}$: {$I_3$, $I_5$}. As $I_2$ has maximum affinity of 60 with $f_2^{oh}$, which is greater than maximum affinity bond 40, so

$I_2$ moves to $f_2^{oh}$ to generate new fragments. So, the new fragments of class City are $f_1^h$:{$I_1$, $I_4$} and $f_2^h$:{$I_2$, $I_3$, $I_5$}.

Then, it generates derived fragments of all member classes of the class using all of the instance objects. For class State, based on primary fragments of class City $f_1^h$ and $f_2^h$, it generates derived fragments $f_1^d$:{$I_1$, $I_3$} and $f_2^d$:{$I_1$, $I_2$, $I_3$}. For class Country, based on the derived fragments of class State, it generates derived fragments $f_1^d$:{$I_1$, $I_2$} and $f_2^d$:{ $I_1$, $I_2$}.

Now, classes have their existing fragments $f^{oh}$ (figure 3.2.9) and sets of derived fragments. Then, it merges these derived fragments with the existing horizontal fragments that has maximum affinity measure aff($f^d_i$, $f^{oh}_k$) with. For example, fragments $f_1^d$ and $f_2^d$ of class State have maximum affinity of 0 and 20 with $f_1^h$ respectively. After merging these derived fragments with existing fragment, the incremental horizontal fragment for class State is $f_1^h$:{$I_1$, $I_2$, $I_3$}. Fragments $f_1^d$ and $f_2^d$ of class Country have maximum affinity of 15 and 15 with $f_1^h$ respectively. After merging these derived fragment with existing fragment, the incremental horizontal fragment of class Country is $f_1^h$:{$I_1$, $I_2$}.

The following table shows incremental horizontal fragments of classes, after changing query access frequencies.

| **Classes** | **Fragments using IOHF Algorithm** | | |
|---|---|---|---|
| *City* | $f_1^h$:{$I_1$, $I_4$} | $f_2^h$:{$I_2$,$I_3$,$I_5$} | |
| *State* | $f_1^h$: {$I_1$,$I_2$,$I_3$} | | |
| *Country* | $f_1^h$:{$I_1$,$I_2$} | | |

**Table 3.2.1.2: Fragments of classes – Changes in Query Access Frequencies**

### *3.2.3 Changes in class hierarchy*

After adding new classes in class inheritance and composition hierarchy, IOHF algorithm generates new super classes/containing classes, which are member classes of these classes. It also generates new subclasses/contained classes of these classes, which are owner classes of these classes. For example, figure 3.2.3.1 shows the class inheritance hierarchy after adding a new class Zone, which is a subclass of class City, and super class of class School and class Hospital. Figure 3.2.3.2 shows the composition hierarchy of classes after adding the class Zone. The object base schema for class Zone is shown in figure 3.2.3.3. Here, CiID, ZoID and ZoName are attributes of class Zone. Again, GetName is the method of the class Zone. Now, instances of

class School and class Hospital point to the instances of class Zone. The changes of class School and class Hospital are shown in figure 3.2.3.3. For example, the instance $I_1$ of class School points to the instance of $I_1$ of class Zone instead of pointing instance $I_1$ of class City. The application queries for new class Zone are shown in figure 3.2.3.4.

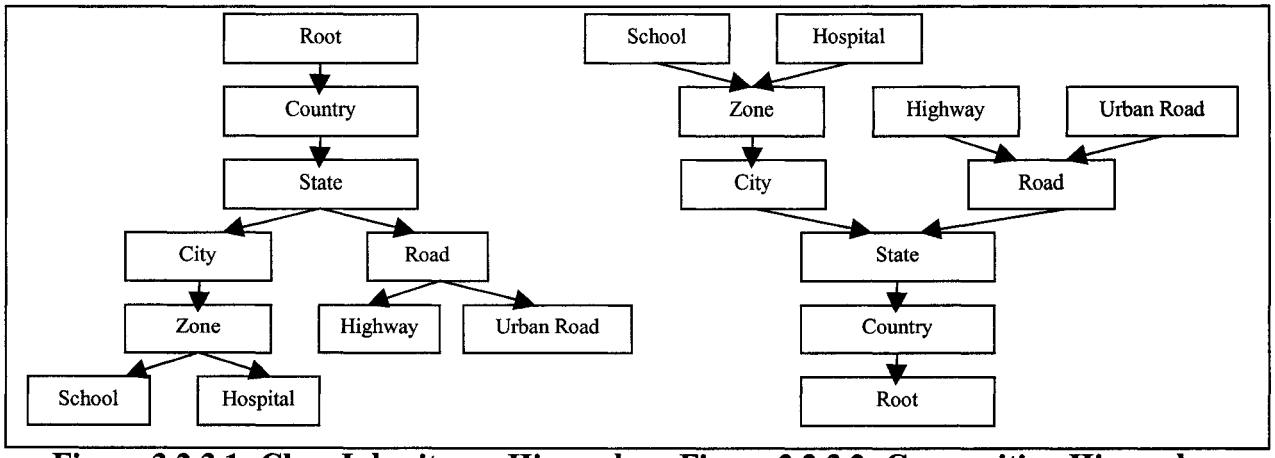


**Figure 3.2.3.1: Class Inheritance Hierarchy – Adding New Class**

**Figure 3.2.3.2: Composition Hierarchy – Adding New Class**

**Zone {CiID, ZoID, ZoName, GetName}**

$I_1$ {City Pointer1, Zo01, Central}
$I_2$ {City Pointer1, Zo02, North}
$I_3$ {City Pointer2, Zo03, West}
$I_4$ {City Pointer2, Zo04, South}
$I_5$ {City Pointer4, Zo05, East}
$I_6$ {City Pointer5, Zo06, East}
$I_7$ {City Pointer3, Zo07, West}

**School {CiID, ScID, ScName, ScStudent, ScCategory, GetName, GetStudent, GetCategory}**

$I_1$ {Zone Pointer1, Sc01, U of T, 4500, University
$I_2$ {Zone Pointer2, Sc02, York University, 4000, University}
$I_3$ {Zone Pointer3, Sc03, U of Windsor, 3000, University}
$I_4$ {Zone Pointer4, Sc04, ABC School, 500, Elementary}
$I_5$ {Zone Pointer5, Sc05, NY University, 3500, University}
$I_6$ {Zone Pointer6, Sc06, M School, 400, Elementary}

**Hospital {CiID, HoID, HoName, HoBeds GetName, GetBeds}**

$I_1$ {Zone Pointer2, Ho01, Toronto Hospital, 200}
$I_2$ {Zone Pointer3, Ho02, Windsor Hospital, 300}
$I_3$ {Zone Pointer4, Ho03, Windsor S Hospital, 150}
$I_4$ {Zone Pointer7, Ho04, Montreal Hospital, 100}
$I_5$ {Zone Pointer5, Ho05, NY Central Hospital, 300}
$I_6$ {Zone Pointer6, Ho06, Q Central Hospital, 150}

**Figure 3.2.3.3: Sample Object Database Schema – Adding New Class**

| |
|---|
| **Query:** Group class *Zone* according to their City name. Query requires method GetName of class Zone and predicates from this query are: |
| $P_1$: CiName = "Toronto" $P_2$: CiName = "Windsor" $P_3$: CiName = "Montreal" |
| $P_4$: CiName = "New York" $P_5$: CiName = "Queens" |

**Figure 3.2.3.4: Application queries – Adding New Class**

Now, applying the minterm predicates of these queries, it generates primary fragments for class Zone. These are $f_1^p$:{$I_1$, $I_2$}, $f_2^p$:{$I_3$, $I_4$}, $f_3^p$:{$I_7$}, $f_4^p$:{$I_5$} and $f_5^p$:{$I_6$}. Based on the existing fragments of class School, it generates derived fragments $f_1^d$: {$I_1$, $I_2$, $I_3$, $I_5$} and $f_2^d$: {$I_4$, $I_6$} for class Zone. Again, based on the existing fragments of class Hospital, it generates derived fragments $f_3^d$: {$I_2$, $I_3$, $I_5$} and $f_4^d$: {$I_4$, $I_6$, $I_7$} for class Zone. Then, for each member class of the class Zone, it obtains derived fragments. For example, the derived fragments of class City, based on primary fragments of class Zone are $f_1^d$:{$I_1$}, $f_2^d$:{$I_2$}, $f_3^d$:{$I_3$}, $f_4^d$:{$I_4$} and $f_5^d$:{$I_5$} and based on the derived fragments of class Zone are $f_6^d$: {$I_1$, $I_2$, $I_4$}, $f_7^d$: {$I_2$, $I_5$}, $f_8^d$: {$I_1$, $I_2$, $I_4$} and $f_9^d$: {$I_2$, $I_3$, $I_5$}. The derived fragments of class State, based on the derived fragments of class City are $f_1^d$:{$I_1$}, $f_2^d$:{$I_1$}, $f_3^d$:{$I_2$}, $f_4^d$:{$I_3$}, $f_5^d$:{$I_3$}, $f_6^d$:{$I_1$, $I_3$}, $f_7^d$:{$I_1$, $I_3$}, $f_8^d$: {$I_1$, $I_3$} and $f_9^d$: {$I_1$, $I_2$}. The derived fragments of class Country, based on the derived fragments of class State are $f_1^d$:{$I_1$}, $f_2^d$:{$I_1$}, $f_3^d$:{$I_1$}, $f_4^d$:{$I_2$}, $f_5^d$:{$I_2$}, $f_6^d$:{$I_1$, $I_2$}, $f_7^d$:{$I_1$, $I_2$}, $f_8^d$: {$I_1$, $I_2$} and $f_9^d$: {$I_1$}.

Assuming access frequencies of class Zone are acco($I_1$) = 10, acco($I_2$) = 10, acco($I_3$) = 10, acco($I_4$) = 5, acco($I_5$) = 10, acco($I_6$) = 10 and acco($I_7$) = 5. Then, it merges the derived fragments with the primary fragments of new classes and the derived fragments with the existing fragments of other classes that has maximum affinity measure aff($F^T_i$, $F^H_k$) with. For example, the derived fragments $f_1^d$, $f_2^d$, $f_3^d$ and $f_4^d$ of new class Zone have maximum affinity of 0, 5, -10 and 0 with $f_1^p$, $f_5^p$, $f_4^p$ and $f_5^p$ respectively. After merging these derived fragments with corresponding primary fragments, we get $f_1^h$:{$I_1$, $I_2$, $I_3$, $I_5$}, $f_2^h$:{$I_3$, $I_4$}, $f_3^h$:{$I_7$}, $f_4^h$:{$I_2$, $I_3$, $I_5$} and $f_5^h$:{$I_4$, $I_6$, $I_7$}. Here, $I_2$, $I_3$, $I_4$, $I_5$, and $I_7$ are overlapping in different fragments. Applying affinity rule 2.2, $I_2$, $I_3$, $I_4$, $I_5$, and $I_7$ are placed in $f_1^h$, $f_1^h$, $f_5^h$, $f_1^h$ and $f_5^h$ with maximum affinity 60, 60, 25, 60 and 25 respectively. Hence, the incremental fragments are $f_1^h$:{$I_1$, $I_2$, $I_3$, $I_5$} and $f_2^h$:{$I_4$, $I_6$, $I_7$}. The derived fragments of class City $f_1^d$, $f_2^d$, $f_3^d$, $f_4^d$, $f_5^d$, $f_6^d$, $f_7^d$, $f_8^d$ and $f_9^d$ have maximum affinity of –5, -15, 0, -5, 0, 25, -5, 25 and 0 with existing fragments $f_1^{oh}$, $f_1^{oh}$, $f_2^{oh}$, $f_1^{oh}$, $f_2^{oh}$, $f_1^{oh}$, $f_2^{oh}$, $f_1^{oh}$ and $f_2^{oh}$ respectively. After merging these derived fragments with corresponding existing fragments, we get $f_1^h$:{$I_1$, $I_2$,

$I_4\}$ and $f_2^h$:$\{I_2, I_3, I_5\}$. Here, $I_2$ is overlapping in different fragments. Applying affinity rule 2.2, $I_2$ is placed in $f_1^h$ with maximum affinity 30. The incremental horizontal fragments are $f_1^h$:$\{I_1, I_2, I_4\}$ and $f_2^h$:$\{I_3, I_5\}$. The derived fragments of class State $f_1^d$, $f_2^d$, $f_3^d$, $f_4^d$, $f_5^d$, $f_6^d$, $f_7^d$, $f_8^d$ and $f_9^d$ have maximum affinity of –5, -5, -15, -5, -5, 15, 15, 15 and 5 with $f_1^{oh}$ respectively. After merging these derived fragments with existing fragment, we get the incremental horizontal fragment $f_1^h$:$\{I_1, I_2, I_3\}$. The derived fragments of class Country $f_1^d$, $f_2^d$, $f_3^d$, $f_4^d$, $f_5^d$, $f_6^d$, $f_7^d$, $f_8^d$ and $f_9^d$ have maximum affinity of 5, 5, 5, -5, -5, 15, 15, 15 and 5 with $f_1^{oh}$ respectively. After merging these derived fragments with existing fragment, we get the incremental fragment $f_1^h$:$\{I_1, I_2\}$.

The following table shows incremental horizontal fragments of classes, after adding a new class Zone.

| **Classes** | **Fragments using IOHF Algorithm** | |
|---|---|---|
| *Zone* | $f_1^h$:$\{I_1,I_2,I_3,I_5\}$ | $f_2^h$:$\{I_4,I_6,I_7\}$ |
| *City* | $f_1^h$:$\{I_1,I_2,I_4\}$ | $f_2^h$:$\{I_3,I_5\}$ |
| *State* | $f_1^h$: $\{I_1,I_2,I_3\}$ | |
| *Country* | $f_1^h$:$\{I_1,I_2\}$ | |

**Table 3.2.3.1: Fragments of classes – Adding New Class**

After deleting a class from class hierarchy, IOHF algorithm generates derived fragments for each member class of this class based on the existing fragments. For example, after deleting the class School from the class inheritance hierarchy (figure 3.2.5), it generates derived fragments for the class City, class State and class Country. The derived fragments of class City, based on the existing fragments of class School are $f_1^d$:$\{I_1, I_2, I_4\}$ and $f_2^d$:$\{I_2, I_5\}$. Then, the derived fragments of class State, based on the derived fragments of class City are $f_1^d$: $\{I_1, I_3\}$ and $f_2^d$:$\{I_1, I_3\}$. The derived fragments of class Country, based on the derived fragments of class State are $f_1^d$:$\{I_1, I_2\}$ and $f_2^d$:$\{I_1, I_2\}$.

Now, it subtracts the derived access frequencies (figure 3.2.1.8) for instances and merges these fragments with the existing horizontal fragments that has highest affinity measure aff($F^d_i$, $F^H_k$) with. For example, after subtracting the derived access frequencies of class School from the total access frequency of class City, we get access frequencies of instances $I_1$, $I_2$, $I_3$, $I_4$ and $I_5$ are 30, 35, 15, 30 and 15 respectively. Fragments $f_1^d$ and $f_2^d$ have highest affinity of 95 and -35 with

$f_1^{oh}$ and $f_2^{oh}$ respectively. After merging these derived fragments with corresponding existing fragments, we get $f_1^h$:{$I_1$, $I_2$, $I_4$} and $f_2^h$:{$I_2$, $I_3$, $I_5$}. Instance $I_2$ is overlapping in different fragments. Applying affinity rule 2.2, $I_2$ is placed in $f_1^h$ with maximum affinity of 130. Hence, the incremental horizontal fragments are $f_1^h$:{$I_1$, $I_2$, $I_4$} and $f_2^h$:{$I_3$, $I_5$}. Similarly, for class State, after subtracting the derived access frequencies of class City, we get access frequencies of instances $I_1$, $I_2$, and $I_3$ are 30, 20, and 20 respectively. Fragments $f_1^d$ and $f_2^d$ have highest affinity of 30 and 30 with $f_1^{oh}$ respectively. After merging these derived fragments with existing fragment, the incremental fragment is $f_1^h$:{$I_1$, $I_2$, $I_3$}. For class Country, after subtracting the derived access frequencies of class State, we get access frequencies of instances $I_1$, and $I_2$ are 10 and 5 respectively. Fragments $f_1^d$ and $f_2^d$ have highest affinity of 15 and 15 with $f_1^{oh}$ respectively. After merging these derived fragments with existing fragment, the incremental fragment is $f_1^h$:{$I_1$, $I_2$}.

The following table shows incremental horizontal fragments of classes, after deleting the class School.

| **Classes** | **Fragments using IOHF Algorithm** | |
|---|---|---|
| *City* | $f_1^h$:{$I_1$,$I_2$,$I_4$} | $f_2^h$:{$I_3$,$I_5$} |
| *State* | $f_1^h$: {$I_1$,$I_2$,$I_3$} | |
| *Country* | $f_1^h$:{$I_1$,$I_2$} | |

**Table 3.2.3.2: Fragments of classes – Deleting Class**

### *3.2.4 Changes in Object Instances*

When new instances arrive for a class at a site, this algorithm applies all minterm predicates on objects that defined the existing fragments, to have them belong to some primary horizontal fragments of the class. Figure 3.2.4.1 shows new instances and their corresponding access frequencies of class School, class Highway and class City. For instances $I_7$ and $I_8$ of class School, it applies the minterm predicate ($MM_1$: ScCategory = "Univeristy" $\wedge$ ScCategory $\neq$ "Elementary") for query 2 of figure 3.2.7 and generates primary fragments $f_1^p$:{$I_8$}. Again after applying minterm predicate ($MM_2$: ScCategory $\neq$ "Univeristy" $\wedge$ ScCategory = "Elementary") to the instances $I_7$ and $I_8$, it generates $f_2^p$:{$I_7$}. Similarly, it generates primary fragments for class Highway is $f_1^p$:{$I_9$} and generates primary fragments for class City are $f_1^p$:{$I_6$} and $f_2^p$:{$I_7$}.

| | | |
|---|---|---|
| ***New Instances*** | | |
| **Class School:** | $I_7${City Pointer3, Sc0007, HM School, 300, Elementary} | |
| | $I_8${City Pointer2, Sc0008, A University, 4000, University} | |
| **Class Highway:** | $I_9${Road Pointer1, Hi0009, 100, 4} | |
| **Class City:** | $I_6${State Pointer1, Ci0006, London, 65000} | |
| | $I_7${State Pointer3, Ci0007, Manhattan, 60000} | |
| ***Access Frequencies*** | | |
| **Class School:** | $acco(I_7) = 10$ | $acco(I_8) = 5$ |
| **Class Highway:** | $acco(I_9) = 5$ | |
| **Class City:** | $acco(I_6) = 10$ | $acco(I_7) = 5$ |

**Figure 3.2.4.1: New Instances and Access Frequencies – Changes in Object Instances**

Then, it obtains derived fragments of all member classes of the class. For example, based on these primary fragments of class School, it generates the derived fragments of class City are $f_1^d$:{$I_2$} and $f_2^d$:{$I_3$}. It generates the derived fragment for class Road, based on the new fragments of class Highway is $f_1^d$:{$I_1$}. The derived fragments of class State, based on primary fragments of class City are $f_1^d$:{$I_1$} and $f_2^d$:{$I_3$}, based on derived fragments of class City are $f_3^d$:{$I_1$} and $f_4^d$:{$I_2$} and based on the derived fragments of class Road is $f_5^d$:{$I_1$}. It also generates the derived fragments of class Country, based on the derived fragment of class State are $f_1^d$:{$I_1$}, $f_2^d$:{$I_2$}, $f_3^d$:{$I_1$}, $f_4^d$:{$I_1$} and $f_5^d${$I_1$}.

Now, it applies the same minterm predicates to each existing fragment of each class. If this process generates any fragment for a class, then it merges the primary fragments with existing fragment with maximum affinity, otherwise the primary fragments will be the new fragments for the class. For example, for class School after applying the same minterm predicate $MM_1$ to $f_1^{oh}$ and $f_2^{oh}$, it generates $f_1^{tp}$:{$I_1$, $I_2$, $I_3$, $I_5$} and $f_2^{tp}$:{}, so the fragment $f_1^p$:{$I_8$} will merge with $f_1^{oh}$ to create fragment $f_1^h$:{ $I_1$, $I_2$, $I_3$, $I_5$, $I_8$}. Again, the other minterm predicate $MM_2$ for class School is applied to $f_1^{oh}$ and $f_2^{oh}$. Then, it generates $f_3^{tp}$:{} and $f_4^{tp}$:{$I_4$, $I_6$}. So, the fragment $f_2^p$:{$I_7$} will merge with $f_2^{oh}$ to create fragment $f_2^h$:{$I_4$, $I_6$, $I_7$}. Similarly, for class Highway, it merges the new fragment $f_1^p$ with $f_1^{oh}$ to generates $f_1^h$:{$I_2$, $I_3$, $I_4$, $I_5$, $I_6$, $I_7$, $I_9$} and $f_2^h$:{$I_1$, $I_8$}. Again, it merges the new fragments $f_1^p$ and $f_2^p$ with $f_1^{oh}$ and $f_2^{oh}$ respectively to create $f_1^h$:{$I_1$, $I_2$, $I_4$, $I_6$} and $f_2^h$:{$I_3$, $I_5$, $I_7$} for class City.

Then, it merges these derived fragments with the existing or new horizontal fragments that has maximum affinity measure aff($F^T_i$, $F^H_k$) with. For example, for class City, fragments $f_1^d$ and $f_2^d$ have maximum affinity of –20 and -5 with new fragments $f_2^h$ respectively. After merging these derived fragments with new fragments, we get are $f_1^h$:{$I_1$, $I_2$, $I_4$, $I_6$} and $f_2^h$:{$I_2$, $I_3$, $I_5$, $I_7$}. Here, $I_2$ is overlapping in different fragments. After applying affinity rule 2.2, $I_2$ is placed in $f_1^h$ with maximum affinity 45. Hence, the incremental fragments are $f_1^h$:{$I_1$, $I_2$, $I_4$, $I_6$} and $f_2^h$:{ $I_3$, $I_5$, $I_7$}. For class Road, fragment $f_1^d$ has maximum affinity of 0 with existing fragment $f_2^{oh}$. After merging $f_1^d$ with the $f_2^{oh}$ fragment, we get incremental fragments $f_1^h$:{$I_1$, $I_2$, $I_3$, $I_4$, $I_5$} and $f_2^h$:{$I_1$, $I_6$}. Here, $I_1$ is overlapping in different fragments. Applying affinity rule 2.2, $I_1$ is placed in $f_1^h$ with maximum affinity of 60. Hence, the incremental horizontal fragments are $f_1^h$:{$I_1$, $I_2$, $I_3$, $I_4$, $I_5$} and $f_2^h$:{$I_6$}. For class State, fragments $f_1^d$, $f_2^d$, $f_3^d$, $f_4^d$ and $f_5^d$ have maximum affinity of –10, –10, -10, 0 and –10 with existing fragment $f_1^{oh}$ respectively. After merging the derived fragments with existing fragment, the incremental horizontal fragment is $f_1^h$:{$I_1$, $I_2$, $I_3$}. For class Country, fragments $f_1^d$, $f_2^d$, $f_3^d$, $f_4^d$ and $f_5^d$ have maximum affinity of 5, -5, 5, 5 and 5 with existing fragment $f_1^{oh}$ respectively. Hence, the incremental horizontal fragment is $f_1^h$:{$I_1$, $I_2$}.

The following table shows incremental horizontal fragments of classes, after adding new instances.

| **Classes** | **Fragments using IOHF Algorithm** | |
|---|---|---|
| *School* | $f_1^h$:{$I_1$,$I_2$,$I_3$,$I_5$,$I_8$} | $f_2^h$:{$I_4$,$I_6$,$I_7$} |
| *Highway* | $f_1^h$:{$I_2$,$I_3$, $I_4$,$I_5$,$I_6$,$I_7$,$I_9$} | $f_2^h$:{$I_1$,$I_8$} |
| *City* | $f_1^h$:{$I_1$,$I_2$,$I_4$,$I_6$} | $f_2^h$:{$I_3$,$I_5$,$I_7$} |
| *Road* | $f_1^h$:{$I_1$,$I_2$,$I_3$,$I_4$,$I_5$} | $f_2^h$:{$I_6$} |
| *State* | $f_1^h$: {$I_1$,$I_2$,$I_3$} | |
| *Country* | $f_1^h$:{$I_1$,$I_2$} | |

**Table 3.2.4.1: Fragments of classes – Adding New Instances**

When instances of a class are deleted, IOHF algorithm removes the objects instances from the corresponding existing fragments and generates new fragments $f^N$. Existing fragments will be new fragments for those classes, which do not have any instances to delete. For example, when the instance $I_2$ is deleted from the object base schema of class School, then this algorithm removes the instance form $f_1^{oh}$: {$I_1$, $I_2$, $I_3$, $I_5$} to create new fragment $f_1^n$: {$I_1$, $I_3$, $I_5$}. Similarly,

when the instance $I_4$ of class School deleted, it generates new fragment $f_2^n$: $\{I_6\}$. Again, after deleting instance $I_3$ from class City, it generates new fragments for class City $f_1^n$:$\{I_1, I_2, I_4\}$ and $f_2^n$:$\{I_5\}$. Due to the integrity constraint, when instance $I_3$ of class City is deleted, the instance $I_4$ of class Hospital must be deleted. Hence, the new fragments for class Hospital are $f_1^n$:$\{I_1, I_2, I_5\}$ and $f_2^n$:$\{I_3, I_6\}$. The new fragments for the remaining classes (Highway, Road, State and Country) are the same as in figure 3.2.9.

Based on these deleted instances, it obtains derived horizontal fragments of all member classes of the class, using all of their instance objects. For example, the derived fragments of class City based on the deleted instances $I_2$ and $I_4$ of class School are $f_1^d$:$\{I_1\}$ and $f_2^d$:$\{I_2\}$ respectively. Again, the derived fragment of class City based on the deleted instance $I_4$ of class Hospital is $f_3^d$:$\{I_3\}$. The derived fragments of class State based on the derived fragments of class City are $f_1^d$:$\{I_1\}$, $f_2^d$:$\{I_1\}$ and $f_3^d$:$\{I_2\}$. Again, the derived fragments of class Country based on the derived fragments of class State are $f_1^d$:$\{I_1\}$, $f_2^d$:$\{I_1\}$ and $f_3^d$:$\{I_1\}$.

Now, it intersects these derived fragments with new fragments to make sure that super class (member) still has a parent instance of it or not. For example, the derived fragment $f_3^d$:$\{I_3\}$ of class City is generated based on the fragment of class Hospital. We know that, the instance $I_3$ of class City has already been deleted. Hence, after intersecting with $f_1^n$ and $f_2^n$, $f_3^d$ will generates two empty sets. So, $f_3^d$ is not a valid derived fragment of class City.

Then, it subtracts the derived access frequencies from total access frequencies and merges these derived fragments with the new fragments with which it has maximum affinity measure aff($F^T_i$, $F^H_k$). For example, after subtracting the derived access frequencies of $I_1$ and $I_4$ for class School from the total access frequencies of class City, we get access frequencies of instances $I_1$, $I_2$, $I_4$ and $I_5$ are 35, 40, 35 and 25 respectively. Now, fragments $f_1^d$ and $f_2^d$ have maximum affinity of -40 and -30 with $f_1^n$ respectively. After merging these derived fragments with corresponding new fragments, the incremental horizontal fragments of class City are $f_1^h$:$\{I_1, I_2, I_4\}$ and $f_3^h$:$\{I_5\}$. Similarly, for class State, after subtracting the derived access frequencies of instance $I_1$ and $I_2$ for class City from total access frequencies, we get access frequencies of instances $I_1$, $I_2$, and $I_3$ are 30, 25 and 35 respectively. Now, $f_1^d$, $f_2^d$ and $f_3^d$ have maximum affinity of -30, -30 and –40 with $f_1^n$ respectively. After merging these derived fragments with $f_1^n$, the incremental horizontal fragment is $f_1^h$:$\{I_1, I_2, I_3\}$. For class Country, after subtracting the derived access frequency of instance $I_1$ for class State from the total access frequency, we get access

frequencies of instances $I_1$, and $I_2$ are 10 and 10 respectively. Now, $f_1^d$, $f_2^d$ and $f_3^d$ have maximum affinity of 0, 0 and 0 with $f_1^n$. After merging these derived fragments with $f_1^n$, the incremental horizontal fragment is $f_1^h$:{$I_1$, $I_2$}.

The following table shows incremental horizontal fragments of classes, after deleting instances.

| Classes | Fragments using IOHF Algorithm | |
|---|---|---|
| *School* | $f_1^h$:{$I_1$,$I_3$,$I_5$} | $f_2^h$:{$I_6$} |
| *Hospital* | $f_1^h$:{$I_1$,$I_2$, $I_5$} | $f_2^h$:{$I_3$,$I_6$} |
| *City* | $f_1^h$:{$I_1$,$I_2$,$I_4$} | $f_2^h$:{$I_5$} |
| *State* | $f_1^h$: {$I_1$,$I_2$,$I_3$} | |
| *Country* | $f_1^h$:{$I_1$,$I_2$} | |

**Table 3.2.4.2: Fragments of classes – Deleting Instances**

# 3.3 Correctness of IOHF algorithm compared to OOHF algorithm

This section reprocesses and re-fragments the object based system described in figure 3.2.4, based on all of the changes of section 3.2 from scratch using OOHF algorithm. Hence, this section shows the correctness of IOHF algorithm. Figure 3.2.5 and figure 3.2.6 show the class inheritance and composition hierarchy for the object based system respectively. This section uses the access frequencies from figure 3.2.8.

## *3.3.1 Changes in Application Access Patterns*

When application access pattern changes due to addition of queries for classes, this section uses the OOHF algorithm to generate horizontal fragments from scratch. Figure 3.2.7 shows the existing application queries for the object based system. Figure 3.2.1.1 shows the new application queries that are to be added into the system. OOHF generates minterm predicates and then generates primary fragments. For example, the primary fragments for classes are

| | | | |
|---|---|---|---|
| Hospital | $f_1^p$: {$I_1$, $I_2$, $I_5$} | $f_2^p$: {$I_3$, $I_4$, $I_6$} | |
| School | $f_1^p$:{$I_1$, $I_2$, $I_5$} | $f_2^p$:{$I_3$} | $f_3^p$: {$I_4$, $I_6$} |
| Highway | $f_1^p$:{$I_2$, $I_3$, $I_4$, $I_5$, $I_6$, $I_7$} | $f_2^p$:{$I_1$, $I_8$} | |
| Road | $f_1^p$:{$I_1$, $I_2$, $I_3$, $I_4$} | $f_2^p$: {$I_5$, $I_6$} | |

| | | | | | |
|---|---|---|---|---|---|
| City | $f_1^p$:{$I_1$} | $f_2^p$:{$I_3$} | $f_3^p$:{$I_4$} | $f_4^p$:{$I_2$} | $f_5^p$:{$I_5$} |
| State | $f_1^p$:{$I_1$} | $f_2^p$:{$I_2$} | $f_3^p$:{$I_3$} | | |

Now, it generates derived fragments for each member classes of the class. For example, the derived fragments of class City based on the primary fragments of class Hospital are $f_1^d$:{$I_1$, $I_2$, $I_4$} and $f_2^d$:{$I_2$, $I_3$, $I_5$} and based on the primary fragments of class School are $f_3^d$:{$I_1$, $I_4$}, $f_4^d$:{$I_2$} and $f_5^d$: {$I_2$, $I_5$}. The derived fragments of class Road based on the primary fragments of class Highway are $f_1^d$:{$I_1$, $I_2$, $I_3$, $I_4$, $I_5$}and $f_2^d$:{$I_1$, $I_6$}. The derived fragments of class State based on the primary fragments of class City are $f_1^d$:{$I_1$}, $f_2^d$:{$I_2$}, $f_3^d$:{$I_3$}, $f_4^d$:{$I_1$} and $f_5^d$:{$I_3$}, based on the derived fragments of class City are $f_6^d$:{$I_1$, $I_3$}, $f_7^d$:{$I_1$, $I_2$, $I_3$}, $f_8^d$:{$I_1$, $I_3$}, $f_9^d$:{ $I_1$} and $f_{10}^d$:{$I_1$, $I_3$}, based on the primary fragments of class Road are $f_{11}^d$:{$I_1$, $I_2$} and $f_{12}^d$:{$I_3$} and based on the derived fragments of class Road are $f_{13}^d$: {$I_1$, $I_2$, $I_3$} and $f_{14}^d$: {$I_1$, $I_3$}. The derived fragments of class Country, based on primary fragments of class State are $f_1^d$:{$I_1$}, $f_2^d$:{$I_1$}, and $f_3^d$:{$I_2$} and based on derived fragments of class State are $f_4^d$:{$I_1$}, $f_5^d$:{$I_1$}, $f_6^d$: {$I_2$}, $f_7^d$:{$I_1$}, $f_8^d$:{$I_2$}, $f_9^d$:{$I_1$, $I_2$}, $f_{10}^d$:{$I_1$, $I_2$}, $f_{11}^d$:{$I_1$, $I_2$}, $f_{12}^d$:{$I_1$}, $f_{13}^d$:{$I_1$, $I_2$}, $f_{14}^d$:{$I_1$}, $f_{15}^d$:{$I_2$}, $f_{16}^d$:{$I_1$, $I_2$} and $f_{17}^d$: {$I_1$, $I_2$}.

Now, it integrates primary and derived fragments to generate the horizontal fragments. It merges derived fragments with primary fragment that has maximum affinity with. For example, for class City, fragments $f_1^d$, $f_2^d$, $f_3^d$, $f_4^d$, and $f_5^d$ have maximum affinity with $f_1^p$, $f_2^p$, $f_1^p$, $f_4^p$ and $f_4^p$ of -5, -5, 0, 5 and 0 respectively. After merging these derived fragments with corresponding primary fragments, we get $f_1^h$:{$I_1$, $I_2$, $I_4$}, $f_2^h$:{$I_2$, $I_3$, $I_5$}, $f_3^h$:{$I_1$, $I_4$}, $f_4^h$:{$I_2$} and $f_4^h$:{$I_2$, $I_5$}. Here, $I_1$, $I_2$, $I_4$ and $I_5$ are overlapping in different fragments. Applying affinity rule 2.2, $I_1$, $I_2$, $I_4$ and $I_5$ have maximum affinity of 35, 30, 35 and 20 with $f_1^h$, $f_1^h$, $f_1^h$ and $f_2^h$ respectively. Hence, horizontal fragments are: $f_1^h$: {$I_1$, $I_2$, $I_4$} and $f_2^h$: {$I_3$, $I_5$}. For class Road, fragments $f_1^d$ and $f_2^d$ have maximum affinity with $f_1^p$ and $f_2^p$ of 25 and –10 respectively. After merging these derived fragment with corresponding primary fragments, we get $f_1^h$: {$I_1$, $I_2$, $I_3$, $I_4$, $I_5$} and $f_2^h$:{$I_1$, $I_5$, $I_6$}. Here, $I_1$ and $I_5$ are overlapping in different fragments. Applying affinity rule 2.2, $I_1$ and $I_5$ are placed in $f_1^h$ and $f_1^h$ with maximum affinity of 60 and 75 respectively. Hence, the horizontal fragments are $f_1^h$: {$I_1$, $I_2$, $I_3$, $I_4$, $I_5$} and $f_2^h$:{$I_6$}. For class State, fragments $f_1^d$, $f_2^d$, $f_3^d$, $f_4^d$, $f_5^d$, $f_6^d$, $f_7^d$, $f_8^d$, $f_9^d$, $f_{10}^d$, $f_{11}^d$, $f_{12}^d$, $f_{13}^d$, and $f_{14}^d$ have maximum affinity with $f_1^p$, $f_2^p$, $f_3^p$, $f_1^p$, $f_3^p$, $f_1^p$, $f_2^p$, $f_1^p$, $f_1^p$, $f_1^p$, $f_2^p$, $f_3^p$, $f_2^p$ and $f_1^p$ of 5, 10, 5, 5, 5, 0, 0, 0, 5, 0, 5, 5, 0 and 0 respectively. After merging these derived fragments with corresponding primary fragments, we get $f_1^h$:{$I_1$}, $f_2^h$:{$I_2$}, $f_3^h$:{$I_3$},

$f_4^h$:{$I_1$}, $f_5^h$:{$I_3$}, $f_6^h$:{$I_1$, $I_3$}, $f_7^h$:{$I_1$, $I_2$, $I_3$}, $f_8^h$:{$I_1$, $I_3$}, $f_9^h$:{$I_1$}, $f_{10}^h$:{$I_1$, $I_3$}, $f_{11}^h$:{$I_1$, $I_2$}, $f_{12}^h$:{$I_3$}, $f_{13}^h$:{$I_1$, $I_2$, $I_3$}, and $f_{14}^h$:{$I_1$, $I_3$}. Here, instances $I_1$, $I_2$ and $I_3$ are overlapping in different fragments. Applying affinity rule 2.2, $I_1$, $I_2$ and $I_3$ are placed in $f_7^h$ with maximum affinity of 25, 30 and 25 respectively. Hence, the horizontal fragment is $f_1^h$: {$I_1$, $I_2$, $I_3$}. As class Country does not have any primary fragments, so the derived fragments will be the horizontal fragments of the class. Here, instances $I_1$, and $I_2$ are overlapping in different fragments. Applying affinity rule 2.2, $I_1$, and $I_2$ are placed in $f_9^d$ with maximum affinity of 15 and 15 respectively. Hence, the horizontal fragment is $f_1^h$: {$I_1$, $I_2$}.

The following table shows fragments generated by IOHF algorithm are identical to the fragments generated by OOHF algorithm for each class, after adding queries.

| **Classes** | **Fragments using OOHF Algorithm** | | | **Fragments using IOHF Algorithm** | | |
|---|---|---|---|---|---|---|
| *School* | $f_1^h$:{$I_1$,$I_2$,$I_5$} | $f_2^h$:{$I_3$} | $f_3^h$:{$I_4$,$I_6$} | $f_1^h$:{$I_1$,$I_2$,$I_5$} | $f_2^h$:{$I_3$} | $f_3^h$:{$I_4$,$I_6$} |
| *Hospital* | $f_1^h$:{$I_1$,$I_2$,$I_5$} | $f_2^h$:{$I_3$,$I_4$,$I_6$} | | $f_1^h$:{$I_1$,$I_2$,$I_5$} | $f_2^h$:{$I_3$,$I_4$,$I_6$} | |
| *Highway* | $f_1^h$:{$I_2$,$I_3$,$I_4$,$I_5$,$I_6$,$I_7$} | $f_2^h$:{$I_1$,$I_8$} | | $f_1^h$:{$I_2$,$I_3$,$I_4$,$I_5$,$I_6$,$I_7$} | $f_2^h$:{$I_1$,$I_8$} | |
| *City* | $f_1^h$:{$I_1$,$I_2$,$I_4$} | $f_2^h$:{$I_3$,$I_5$} | | $f_1^h$:{$I_1$,$I_2$,$I_4$} | $f_2^h$:{$I_3$,$I_5$} | |
| *Road* | $f_1^h$:{$I_1$,$I_2$,$I_3$,$I_4$,$I_5$} | $f_2^h$:{$I_6$} | | $f_1^h$:{$I_1$,$I_2$,$I_3$,$I_4$,$I_5$} | $f_2^h$:{$I_6$} | |
| *State* | $f_1^h$: {$I_1$,$I_2$,$I_3$} | | | $f_1^h$: {$I_1$,$I_2$,$I_3$} | | |
| *Country* | $f_1^h$:{$I_1$,$I_2$} | | | $f_1^h$:{$I_1$,$I_2$} | | |

**Table 3.3.1.1: Fragments of classes – Adding Queries**

When application access pattern changes due to deletion of query 2 and query 3 of figure 3.2.7, OOHF algorithm generates primary fragments of classes. The primary fragments of classes are

| | | | |
|---|---|---|---|
| Hospital | $f_1^p$: {$I_1$, $I_2$, $I_5$} | $f_2^p$: {$I_3$, $I_4$, $I_6$} | |
| Highway | $f_1^p$: {$I_2$, $I_3$, $I_4$, $I_5$, $I_6$, $I_7$} | $f_2^p$: {$I_1$, $I_8$} | |
| State | $f_1^p$:{$I_1$} | $f_2^p$:{$I_2$} | $f_3^p$:{$I_3$} |

Now it generates derived fragments for each member classes of the class. For example, the derived fragments of class City, based on primary fragments of class Hospital are $f_1^d$:{$I_1$, $I_2$, $I_4$} and $f_2^d$:{$I_2$, $I_3$, $I_5$}. The derived fragments of class Road, based on the primary fragments of class Highway are $f_1^d$:{$I_1$, $I_2$, $I_3$, $I_4$, $I_5$} and $f_2^d$:{$I_1$, $I_6$}. The derived fragments of class State, based on derived fragments of class City are $f_1^d$:{ $I_1$, $I_3$} and $f_2^d$:{ $I_1$, $I_2$, $I_3$}. Again, the derived

fragments of class State, based on the derived fragments of class Road are $f_3^d$:{$I_1$, $I_2$, $I_3$} and $f_4^d$:{$I_1$, $I_3$}. The derived fragments of class Country, based on primary fragments of class State are $f_1^d$:{$I_1$}, $f_2^d$:{$I_1$} and $f_3^d$:{$I_2$}, and based on the derived fragments of class State are $f_4^d$:{$I_1$, $I_2$}, $f_5^d$:{$I_1$, $I_2$}, $f_6^d$:{$I_1$, $I_2$} and $f_7^d$:{$I_1$, $I_2$}.

Now, it integrates primary and derived fragments to generate the horizontal fragments. It merges derived fragments with primary fragment that has maximum affinity with. For example, as class City does not have any primary fragments, so the derived fragments will be the horizontal fragments of the class. Here, $I_2$ is overlapping in different fragments. Applying affinity rule 2.2, $I_2$ is placed in $f_1^h$ with maximum affinity of 30. Hence, the final fragments are $f_1^h$:{$I_1$, $I_2$, $I_4$} and $f_2^h$:{$I_3$, $I_5$}. Again, as class Road does not have any primary fragments, so the derived fragments will be the horizontal fragments of the class. Here, instance $I_1$ is overlapping in different fragments. Applying affinity rule 2.2, $I_1$ is placed in $f_1^h$ with maximum affinity of 60. Hence, the final fragments are $f_1^h$:{$I_1$, $I_2$, $I_3$, $I_4$, $I_5$} and $f_2^h$:{$I_6$}. For class State, fragments $f_1^d$, $f_2^d$, $f_3^d$ and $f_4^d$ have maximum affinity of 0, 0, 0 and 0 with $f_1^p$, $f_2^p$, $f_2^p$ and $f_1^p$ respectively. After merging these derived fragments with corresponding primary fragments, we get $f_1^h$:{$I_1$, $I_3$} and $f_2^h$:{$I_1$, $I_2$, $I_3$}. Here, $I_1$ and $I_3$ are overlapping in different fragments. Applying affinity rule 2.2, $I_1$ and $I_3$ are placed in $f_2^h$ with maximum affinity of 25 and 25 respectively. Hence, the final fragment is $f_1^h$:{$I_1$, $I_2$, $I_3$}. As class Country does not have any primary fragments, so the derived fragments will be the horizontal fragments. Here, $I_1$ and $I_2$ are overlapping in different fragments. Applying affinity rule 2.2, $I_1$ and $I_2$ have maximum affinity of 15 and 15 with $f_4^d$. Hence, the final fragment for country is $f_1^h$:{$I_1$, $I_2$}.

The following table shows fragments generated by IOHF algorithm are identical to the fragments generated by OOHF algorithm for each class, after deleting queries.

| Classes | Fragments using OOHF Algorithm | | Fragments using IOHF Algorithm | |
|---|---|---|---|---|
| *School* | $f_1^h$:{$I_1,I_2,I_3,I_4,I_5,I_6$} | | $f_1^h$:{$I_1,I_2,I_5,I_3,I_4,I_6$} | |
| *Hospital* | $f_1^h$:{$I_1,I_2,I_5$} | $f_2^h$:{$I_3,I_4,I_6$} | $f_1^h$:{$I_1,I_2,I_5$} | $f_2^h$:{$I_3,I_4,I_6$} |
| *Highway* | $f_1^h$:{$I_2,I_3,I_4,I_5,I_6,I_7$} | $f_2^h$:{$I_1,I_8$} | $f_1^h$:{$I_2,I_3,I_4,I_5,I_6,I_7$} | $f_2^h$:{$I_1,I_8$} |
| *City* | $f_1^h$:{$I_1,I_2,I_4$} | $f_2^h$:{$I_3,I_5$} | $f_1^h$:{$I_1,I_2,I_4$} | $f_2^h$:{$I_3,I_5$} |
| *Road* | $f_1^h$:{$I_1,I_2,I_3,I_4,I_5$} | $f_2^h$:{$I_6$} | $f_1^h$:{$I_1,I_2,I_3,I_4,I_5$} | $f_2^h$:{$I_6$} |
| *State* | $f_1^h$: {$I_1,I_2,I_3$} | | $f_1^h$: {$I_1,I_2,I_3$} | |
| *Country* | $f_1^h$:{$I_1,I_2$} | | $f_1^h$:{$I_1,I_2$} | |

**Table 3.3.1.2: Fragments of classes – Deleting Queries**

### *3.3.2 Changes in Application Query Access Frequencies*

When application access frequencies changes, this section uses the OOHF algorithm to generate horizontal fragments from scratch. For example, assuming the new access frequencies of instances of class City are acco($I_1$) = 5, acco($I_2$) = 15, acco($I_3$) = 15, acco($I_4$) = 5 and acco($I_5$) = 15, which are same as access frequencies of section 3.2.2. The access frequencies for all instances of other classes are same as in figure 3.2.8.

Now, it generates minterm predicates and primary fragments for each class of the class inheritance hierarchy (figure 3.2.5) based on the application queries from figure 3.2.7. The primary fragments for classes are

| | | | |
|---|---|---|---|
| Hospital | $f_1^p$: {$I_1$, $I_2$, $I_5$} | $f_2^p$: {$I_3$, $I_4$, $I_6$} | |
| School | $f_1^p$:{$I_1$, $I_2$, $I_3$, $I_5$} | $f_2^p$: {$I_4$, $I_6$} | |
| Highway | $f_1^p$: {$I_2$, $I_3$, $I_4$, $I_5$, $I_6$, $I_7$} | $f_2^p$: {$I_1$, $I_8$} | |
| City | $f_1^p$:{$I_1$, $I_4$} | $f_2^p$:{$I_2$, $I_3$, $I_5$} | |
| State | $f_1^p$: {$I_1$} | $f_2^p$: {$I_2$} | $f_2^p$: {$I_3$}. |

Then, it generates derived fragments for each member classes of the class. For example, the derived fragments of class City, based on primary fragments of class Hospital are $f_1^d$:{ $I_1$, $I_2$, $I_4$} and $f_2^d$: { $I_2$, $I_3$, $I_5$}, and based on primary fragments of class School are $f_3^d$:{$I_1$, $I_2$, $I_4$} and $f_4^d$: { $I_2$, $I_5$}. The derived fragments of class Road, based on the primary fragments of class Highway are $f_1^d$:{$I_1$, $I_2$, $I_3$, $I_4$, $I_5$} and $f_2^d$:{$I_1$, $I_6$}. The derived fragments of class State, based on primary

fragments of class City are $f_1^d$:{$I_1$, $I_3$} and $f_2^d$:{$I_1$, $I_2$, $I_3$}, based on the derived fragments of class City are $f_3^d$:{$I_1$, $I_3$}, $f_4^d$:{$I_1$, $I_2$, $I_3$}, $f_5^d$:{$I_1$, $I_3$} and $f_6^d$: {$I_1$, $I_3$}, and based on the derived fragments of class Road are $f_7^d$:{$I_1$, $I_2$, $I_3$} and $f_8^d$:{$I_1$, $I_3$}. The derived fragments of class Country, based on the primary fragments of class State are $f_1^d$:{$I_1$}, $f_2^d$:{$I_1$} and $f_3^d$:{$I_2$} and based on derived fragments of class State are $f_4^d$:{$I_1$, $I_2$}, $f_5^d$:{$I_1$, $I_2$}, $f_6^d$:{$I_1$, $I_2$}, $f_7^d$:{$I_1$, $I_2$}, $f_8^d$:{$I_1$, $I_2$}, $f_9^d$:{$I_1$, $I_2$}, $f_{10}^d$:{$I_1$, $I_2$} and $f_{11}^d$:{$I_1$, $I_2$}.

Now, it integrates primary and derived fragments to generate the horizontal fragments. It merges derived fragments with primary fragment that has maximum affinity with. For example, for class City $f_1^d$, $f_2^d$, $f_3^d$, and $f_4^d$ have the maximum affinity with $f_1^p$, $f_2^p$, $f_1^p$ and $f_2^p$ of -5, 45, -5 and 15 respectively. After merging these derived fragments with corresponding primary fragments, we get $f_1^h$:{$I_1$, $I_2$, $I_4$}, and $f_2^h$:{$I_2$, $I_3$, $I_5$}. Here, $I_2$ is overlapping in different fragments. Applying affinity rule 2.2, $I_2$ is placed in $f_2^h$ with maximum affinity of 60. The horizontal fragmentation are: $f_1^h$:{$I_1$, $I_4$}, and $f_2^h$:{$I_2$, $I_3$, $I_5$}. As class Road does not have any primary fragments, so the derived fragments will be the horizontal fragments of the class. Here, $I_1$ overlaps in different fragments. Applying affinity rule 2.2, $I_1$ is placed in $f_1^d$ with maximum affinity 60. Hence, the final horizontal fragments are $f_1^h$: {$I_1$, $I_2$, $I_3$, $I_4$, $I_5$} and $f_2^h$: {$I_6$}. For class State, fragments $f_1^d$, $f_2^d$, $f_3^d$, $f_4^d$, $f_5^d$, $f_6^d$, $f_7^d$, and $f_8^d$ have maximum affinity with $f_1^p$, $f_2^p$, $f_1^p$, $f_2^p$, $f_1^p$, $f_1^p$, $f_2^p$, and $f_1^p$ of 0, 0, 0, 0, 0, 0, 0 and 0 respectively. After merging these derived fragments with corresponding primary fragments, we get $f_1^h$:{$I_1$, $I_3$}, $f_2^h$:{$I_1$, $I_2$, $I_3$}, $f_3^h$:{$I_1$, $I_3$}, $f_4^h$:{ $I_1$, $I_2$, $I_3$}, $f_5^h$:{ $I_1$, $I_3$}, $f_6^h$:{$I_1$, $I_3$}, $f_7^h$:{ $I_1$, $I_2$, $I_3$} and $f_8^h$:{$I_1$, $I_3$}. Here, instances $I_1$, $I_2$ and $I_3$ are overlapping in different fragments. Applying affinity rule 2.2, $I_1$, $I_2$ and $I_3$ are placed in $f_2^h$ with maximum affinity of 25, 30 and 25 respectively. Hence, the horizontal fragment is $f_1^h$: {$I_1$, $I_2$, $I_3$}. As class Country does not have any primary fragments, so the derived fragments will be the horizontal fragments of the class. Here, $I_1$ and $I_2$ are overlapping in different fragments. Applying affinity rule 2.2, $I_1$ and $I_2$ are placed in $f_4^d$ with maximum affinity of 15 and 15 respectively. Hence, the final horizontal fragment is $f_1^h$: {$I_1$, $I_2$}.

The following table shows fragments generated by IOHF algorithm are identical to the fragments generated by OOHF algorithm for each class, after changing query access frequencies.

| Classes | Fragments using OOHF Algorithm | | Fragments using IOHF Algorithm | |
|---|---|---|---|---|
| *School* | $f_1^h$:{$I_1$,$I_2$,$I_3$,$I_5$} | $f_1^h$:{$I_4$,$I_6$} | $f_1^h$:{$I_1$,$I_2$,$I_3$,$I_5$} | $f_1^h$:{$I_4$,$I_6$} |
| *Hospital* | $f_1^h$:{$I_1$,$I_2$,$I_5$} | $f_2^h$:{$I_3$,$I_4$,$I_6$} | $f_1^h$:{$I_1$,$I_2$,$I_5$} | $f_2^h$:{$I_3$,$I_4$,$I_6$} |
| *Highway* | $f_1^h$:{$I_2$,$I_3$,$I_4$,$I_5$,$I_6$,$I_7$} | $f_2^h$:{$I_1$,$I_8$} | $f_1^h$:{$I_2$,$I_3$,$I_4$,$I_5$,$I_6$,$I_7$} | $f_2^h$:{$I_1$,$I_8$} |
| *City* | $f_1^h$:{$I_1$,$I_4$} | $f_2^h$:{$I_2$,$I_3$,$I_5$} | $f_1^h$:{$I_1$,$I_4$} | $f_2^h$:{$I_2$,$I_3$,$I_5$} |
| *Road* | $f_1^h$:{$I_1$,$I_2$,$I_3$,$I_4$,$I_5$} | $f_2^h$:{$I_6$} | $f_1^h$:{$I_1$,$I_2$,$I_3$,$I_4$,$I_5$} | $f_2^h$:{$I_6$} |
| *State* | $f_1^h$: {$I_1$,$I_2$,$I_3$} | | $f_1^h$: {$I_1$,$I_2$,$I_3$} | |
| *Country* | $f_1^h$:{$I_1$,$I_2$} | | $f_1^h$:{$I_1$,$I_2$} | |

**Table 3.3.2.1: Fragments of classes – Changes in Query Access Frequencies**

### *3.3.3 Changes in class hierarchy*

This section uses the OOHF algorithm to generate horizontal fragments from scratch, when new classes are added into class hierarchy. It uses class inheritance and composition hierarchy from figure 3.2.3.1 and 3.2.3.2 respectively. It also uses the sample data for class Hospital, School and Zone from figure 3.2.3.3 and for class Highway, Urbanroad, Road, City, State and Country from figure 3.2.4.

Now, it generates minterm predicates and primary fragments for the classes School, Hospital, Highway, City and State based on the application queries of figure 3.2.7, these are

| | | | |
|---|---|---|---|
| Hospital | $f_1^p$:{$I_1$, $I_2$, $I_5$} | $f_2^p$:{$I_3$, $I_4$, $I_6$} | |
| School | $f_1^p$:{$I_1$, $I_2$, $I_3$, $I_5$} | $f_2^p$:{$I_4$, $I_6$} | |
| Highway | $f_1^p$:{$I_2$, $I_3$, $I_4$, $I_5$, $I_6$, $I_7$} | $f_2^p$:{$I_1$, $I_8$} | |
| City | $f_1^p$:{$I_1$, $I_4$} | $f_2^p$:{$I_2$, $I_3$, $I_5$} | |
| State | $f_1^p$:{$I_1$} | $f_2^p$:{$I_2$} | $f_2^p$:{$I_3$} |

It also generates minterm predicates and primary fragments for class Zone, based on the application queries of figure 3.2.3.4. The primary fragments of class Zone are

$f_1^p$:{$I_1$, $I_2$} $f_2^p$:{$I_3$, $I_4$} $f_3^p$:{$I_7$} $f_4^p$:{$I_5$} $f_5^p$:{$I_6$}

Then, it generates derived fragments for each member classes of the class. For example, the derived fragments of class Zone, based on primary fragments of class School are $f_1^d$: {$I_1$, $I_2$, $I_3$, $I_5$} and $f_2^d$: {$I_4$, $I_6$}, and based on primary fragments of class Hospital are $f_3^d$: {$I_2$, $I_3$, $I_5$} and

$f_4^d$: $\{I_4, I_6, I_7\}$. The derived fragments of class City, based on the primary fragments of class Zone are $f_1^d$:$\{I_1\}$, $f_2^d$:$\{I_2\}$, $f_3^d$:$\{I_3\}$, $f_4^d$:$\{I_4\}$ and $f_2^d$:$\{I_5\}$ and based on the derived fragments of class Zone are $f_6^d$: $\{I_1, I_2, I_4\}$, $f_7^d$: $\{I_2, I_5\}$, $f_8^d$: $\{I_1, I_2, I_4\}$ and $f_9^d$: $\{I_2, I_3, I_5\}$. The derived fragments of class Road based on the primary fragments of class Highway are $f_1^d$:$\{I_1, I_2, I_3, I_4, I_5\}$ and $f_2^d$:$\{ I_1, I_6\}$. The derived fragments of class State, based on primary fragments of class City are $f_1^d$:$\{I_1, I_3\}$ and $f_2^d$:$\{I_1, I_2, I_3\}$, based on the derived fragments of class City are: $f_3^d$:$\{I_1\}$, $f_4^d$:$\{I_1\}$, $f_5^d$:$\{I_2\}$, $f_6^d$:$\{I_3\}$, $f_7^d$:$\{I_3\}$, $f_8^d$:$\{I_1, I_3\}$, $f_9^d$:$\{I_1, I_3\}$, $f_{10}^d$:$\{I_1, I_3\}$ and $f_{11}^d$:$\{I_1, I_2, I_3\}$ and based on the derived fragments of owner class Road are $f_{12}^d$:$\{I_1, I_2, I_3\}$ and $f_{13}^d$:$\{I_1, I_3\}$. The derived fragments of class Country, based on primary fragments of class State are $f_1^d$:$\{I_1\}$, $f_2^d$:$\{I_1\}$ and $f_3^d$: $\{I_2\}$ and based on the derived fragments of class State are $f_4^d$:$\{I_1, I_2\}$ and $f_5^d$:$\{I_1, I_2\}$, $f_6^d$:$\{I_1\}$, $f_7^d$:$\{I_1\}$, $f_8^d$:$\{I_1\}$, $f_9^d$:$\{I_2\}$, $f_{10}^d$:$\{I_2\}$, $f_{11}^d$:$\{I_1, I_2\}$, $f_{12}^d$:$\{I_1, I_2\}$, $f_{13}^d$: $\{I_1, I_2\}$, $f_{14}^d$: $\{I_1, I_2\}$, $f_{15}^d$:$\{I_1, I_2\}$ and $f_{16}^d$:$\{I_1, I_2\}$.

Now, it integrates primary and derived fragments to generate the horizontal fragments. It merges derived fragments with primary fragment that has maximum affinity with. For example, for class Zone $f_1^d$, $f_2^d$, $f_3^d$ and $f_4^d$ have maximum affinity of 0, 5, -10 and 0 with $f_1^p$, $f_5^p$, $f_4^p$ and $f_5^p$ respectively. After merging these derived fragments with the corresponding primary fragments, we get $f_1^h$:$\{I_1, I_2, I_3, I_5\}$, $f_2^h$:$\{I_3, I_4\}$, $f_3^h$:$\{I_7\}$, $f_4^h$:$\{I_2, I_3, I_5\}$ and $f_5^h$:$\{I_4, I_6, I_7\}$. Here, $I_2$, $I_3$, $I_4$, $I_5$, and $I_7$ are overlapping in different fragments. Applying affinity rule 2.2, $I_2$, $I_3$, $I_4$, $I_5$, and $I_7$ are placed in $f_1^h$, $f_1^h$, $f_5^h$, $f_1^h$ and $f_5^h$ with maximum affinity 60, 60, 25, 60 and 25 respectively. The final fragments are $f_1^h$:$\{I_1, I_2, I_3, I_5\}$ and $f_2^h$:$\{I_4, I_6, I_7\}$. For class City $f_1^d$, $f_2^d$, $f_3^d$, $f_4^d$, $f_5^d$, $f_6^d$, $f_7^d$, $f_8^d$ and $f_9^d$ have maximum affinity of 0, -5, -5, 0, -5, 15, 5, 15 and 15 with $f_1^p$, $f_2^p$, $f_2^p$, $f_1^p$, $f_2^p$, $f_1^p$, $f_2^p$, $f_1^p$ and $f_2^p$ respectively. After merging these derived fragments with corresponding primary fragments, we get $f_1^h$:$\{I_1, I_2, I_4\}$ and $f_2^h$: $\{I_2, I_3, I_5\}$. Here, $I_2$ is overlapping in different fragments. Applying affinity rule 2.2, $I_2$ is placed in $f_1^h$ with maximum affinity of 30. Hence, the final fragments are $f_1^h$:$\{I_1, I_2, I_4\}$ and $f_2^h$:$\{I_3, I_5\}$. As class Road does not have any primary fragments, so the derived fragments will be the horizontal fragments of the class. Here, $I_1$ overlaps in different fragments. Applying affinity rule 2.2, $I_1$ is placed in $f_1^d$ with maximum affinity 60. Hence, the final horizontal fragments are $f_1^h$: $\{I_1, I_2, I_3, I_4, I_5\}$ and $f_2^h$: $\{I_6\}$. For class State, fragments $f_1^d$, $f_2^d$, $f_3^d$, $f_4^d$, $f_5^d$, $f_6^d$, $f_7^d$, $f_8^d$, $f_9^d$, $f_{10}^d$, $f_{11}^d$, $f_{12}^d$, and $f_{13}^d$ have maximum affinity with $f_1^p$, $f_2^p$, $f_1^p$, $f_1^p$, $f_2^p$, $f_3^p$, $f_3^p$, $f_1^p$, $f_1^p$, $f_1^p$, $f_2^p$, $f_2^p$ and $f_1^p$ of 0, 0, 5, 5, 10, 5, 5, 0, 0, 0, 0, 0 and 5 respectively. After merging these derived fragments with corresponding primary fragments, we

get $f_1^h$:{ $I_1$, $I_3$}, $f_2^h$:{ $I_1$, $I_2$, $I_3$}, $f_3^h$:{$I_1$}, $f_4^h$:{$I_1$}, $f_5^h$:{$I_2$}, $f_6^h$:{$I_3$}, $f_7^h$:{$I_3$}, $f_8^h$:{$I_1$, $I_3$}, $f_9^h$:{$I_1$, $I_3$}, $f_{10}^h$:{$I_1$, $I_3$}, $f_{11}^h$:{$I_1$, $I_2$, $I_3$}, $f_{12}^h$:{ $I_1$, $I_2$, $I_3$} and $f_{13}^h$:{$I_1$, $I_3$}. Here, instances $I_1$, $I_2$ and $I_3$ are overlapping in different fragments. Applying affinity rule 2.2, $I_1$, $I_2$ and $I_3$ are placed in $f_2^h$ with maximum affinity of 25, 30 and 25 respectively. Hence, the horizontal fragment is $f_1^h$: {$I_1$, $I_2$, $I_3$}. As class Country does not have any primary fragments, so the derived fragments will be the horizontal fragments of the class. Here, $I_1$ and $I_2$ are overlapping in different fragments. Applying affinity rule 2.2, $I_1$ and $I_2$ are placed in $f_4^d$ with maximum affinity of 15 and 15 respectively. Hence, the final fragment is $f_1^h$:{$I_1$, $I_2$}.

The following table shows fragments generated by IOHF algorithm are identical to the fragments generated by OOHF algorithm for each class, after adding class.

| Classes | Fragments using OOHF Algorithm | | Fragments using IOHF Algorithm | |
|---|---|---|---|---|
| *School* | $f_1^h$:{$I_1$,$I_2$,$I_3$,$I_5$} | $f_1^h$:{$I_4$,$I_6$} | $f_1^h$:{$I_1$,$I_2$,$I_3$,$I_5$} | $f_1^h$:{$I_4$,$I_6$} |
| *Hospital* | $f_1^h$:{$I_1$,$I_2$,$I_5$} | $f_2^h$:{$I_3$,$I_4$,$I_6$} | $f_1^h$:{$I_1$,$I_2$,$I_5$} | $f_2^h$:{$I_3$,$I_4$,$I_6$} |
| *Highway* | $f_1^h$:{$I_2$,$I_3$,$I_4$,$I_5$,$I_6$,$I_7$} | $f_2^h$:{$I_1$,$I_8$} | $f_1^h$:{$I_2$,$I_3$,$I_4$,$I_5$,$I_6$,$I_7$} | $f_2^h$:{$I_1$,$I_8$} |
| *Zone* | $f_1^h$:{$I_1$,$I_2$,$I_3$,$I_5$} | $f_2^h$:{$I_4$,$I_6$,$I_7$} | $f_1^h$:{$I_1$,$I_2$,$I_3$,$I_5$} | $f_2^h$:{$I_4$,$I_6$,$I_7$} |
| *City* | $f_1^h$:{$I_1$,$I_2$,$I_4$} | $f_2^h$:{$I_3$,$I_5$} | $f_1^h$:{$I_1$,$I_2$,$I_4$} | $f_2^h$:{$I_3$,$I_5$} |
| *Road* | $f_1^h$:{$I_1$,$I_2$,$I_3$,$I_4$,$I_5$} | $f_2^h$:{$I_6$} | $f_1^h$:{$I_1$,$I_2$,$I_3$,$I_4$,$I_5$} | $f_2^h$:{$I_6$} |
| *State* | $f_1^h$: {$I_1$,$I_2$,$I_3$} | | $f_1^h$: {$I_1$,$I_2$,$I_3$} | |
| *Country* | $f_1^h$:{$I_1$,$I_2$} | | $f_1^h$:{$I_1$,$I_2$} | |

**Table 3.3.3.1: Fragments of classes – Adding Class**

After deleting a class e.g., class School, from the class inheritance hierarchy of figure 3.2.5, OOHF algorithm generates minterm predicates and primary fragments for all other classes (Hospital, Highway, City and State) based on application queries of figure 3.2.7. The primary fragments of these classes are

| | | | |
|---|---|---|---|
| Hospital | $f_1^p$: {$I_1$, $I_2$, $I_5$} | $f_2^p$: {$I_3$, $I_4$, $I_6$} | |
| Highway | $f_1^p$: {$I_2$, $I_3$, $I_4$, $I_5$, $I_6$, $I_7$} | $f_2^p$: {$I_1$, $I_8$} | |
| City | $f_1^p$:{$I_1$, $I_4$} | $f_2^p$:{$I_2$, $I_3$, $I_5$} | |
| State | $f_1^p$: {$I_1$} | $f_2^p$: {$I_2$} | $f_2^p$: {$I_3$}. |

Then, it generates derived fragments for each member classes of the class. For example, the derived fragments of class City, based on primary fragments of class Hospital are $f_1^d$:{$I_1$, $I_2$, $I_4$} and $f_2^d$:{$I_2$, $I_3$, $I_5$}. The derived fragments of class Road, based on primary fragments of class

Highway are $f_1^d$:{$I_1$, $I_2$, $I_3$, $I_4$, $I_5$} and $f_2^d$:{$I_1$, $I_6$}. Derived fragments of class State based on primary fragments of class City are $f_1^d$:{ $I_1$, $I_3$} and $f_2^d$:{ $I_1$, $I_2$, $I_3$}, based on derived fragments of class City are $f_3^d$:{ $I_1$, $I_3$} and $f_4^d$:{ $I_1$, $I_2$, $I_3$} and based on the derived fragments of class Road are $f_5^d$:{$I_1$, $I_2$, $I_3$} and $f_6^d$:{$I_1$, $I_3$}. The derived fragments of class Country, based on primary fragments of class State are $f_1^d$:{$I_1$}, $f_2^d$:{$I_1$} and $f_3^d$:{$I_2$} and based on derived fragments of class State are $f_4^d$: {$I_1$, $I_2$}, $f_5^d$: {$I_1$, $I_2$}, $f_6^d$: {$I_1$, $I_2$}, $f_7^d$: {$I_1$, $I_2$}, $f_8^d$:{$I_1$, $I_2$} and $f_9^d$:{$I_1$, $I_2$}.

Now, it integrates primary and derived fragments to generate the horizontal fragments. It merges derived fragments with primary fragment that has maximum affinity with. For example, for class City, $f_1^d$ and $f_2^d$ have maximum affinity of 15 and 15 with $f_1^p$ and $f_2^p$ respectively. After merging these derived fragments with corresponding primary fragments, we get $f_1^h$:{$I_1$, $I_2$, $I_4$} and $f_2^h$:{$I_2$, $I_3$, $I_5$}. Here, $I_2$ is overlapping in different fragments. Applying affinity rule 2.2, $I_2$ is placed in $f_1^h$ with maximum affinity of 30. Hence, the final fragments are $f_1^h$:{$I_1$, $I_2$, $I_4$} and $f_2^h$:{$I_3$, $I_5$}. As class Road does not have any primary fragments, so the derived fragments will be the horizontal fragments for the class. In the class Road $I_1$ is overlapping in different fragments. Applying affinity rule 2.2, $I_1$ is placed in $f_1^h$ with maximum affinity of 60. Hence, the final fragments are $f_1^h$:{$I_1$, $I_2$, $I_3$, $I_4$, $I_5$} and $f_2^h$:{$I_6$}. For class State, $f_1^d$, $f_2^d$, $f_3^d$, $f_4^d$, $f_5^d$ and $f_6^d$ have maximum affinity of 0, 0, 0, 0, 0 and 0 with $f_1^p$, $f_2^p$, $f_1^p$, $f_2^p$, $f_2^p$ and $f_1^p$ respectively. After merging these derived fragments with corresponding primary fragments, we get $f_1^h$:{$I_1$, $I_3$} and $f_2^h$:{$I_1$, $I_2$, $I_3$}. Here, $I_1$ and $I_3$ are overlapping in different fragments. Applying affinity rule 2.2, $I_1$ and $I_3$ are placed in $f_2^h$ with maximum affinity of 25 and 25 respectively. Hence, the final fragment is $f_1^h$:{$I_1$, $I_2$, $I_3$}. As class Country does not have any primary fragments, so the derived fragments will be the horizontal fragments of the class. Here, $I_1$ and $I_2$ are overlapping in different fragments. Applying affinity rule 2.2, $I_1$ and $I_2$ are placed in $f_4^d$ with maximum affinity of 15 and 15 respectively. Hence, the final fragment is $f_1^h$:{$I_1$, $I_2$}.

The following table shows fragments generated by IOHF algorithm are identical to the fragments generated by OOHF algorithm for each class, after deleting class.

| Classes | Fragments using OOHF Algorithm | | Fragments using IOHF Algorithm | |
|---|---|---|---|---|
| *Hospital* | $f_1^h$:{$I_1$,$I_2$,$I_5$} | $f_2^h$:{$I_3$,$I_4$,$I_6$} | $f_1^h$:{$I_1$,$I_2$,$I_5$} | $f_2^h$:{$I_3$,$I_4$,$I_6$} |
| *Highway* | $f_1^h$:{$I_2$,$I_3$,$I_4$,$I_5$,$I_6$,$I_7$} | $f_2^h$:{$I_1$,$I_8$} | $f_1^h$:{$I_2$,$I_3$,$I_4$,$I_5$,$I_6$,$I_7$} | $f_2^h$:{$I_1$,$I_8$} |
| *City* | $f_1^h$:{$I_1$,$I_2$,$I_4$} | $f_2^h$:{$I_3$,$I_5$} | $f_1^h$:{$I_1$,$I_2$,$I_4$} | $f_2^h$:{$I_3$,$I_5$} |
| *Road* | $f_1^h$:{$I_1$,$I_2$,$I_3$,$I_4$,$I_5$} | $f_2^h$:{$I_6$} | $f_1^h$:{$I_1$,$I_2$,$I_3$,$I_4$,$I_5$} | $f_2^h$:{$I_6$} |
| *State* | $f_1^h$: {$I_1$,$I_2$,$I_3$} | | $f_1^h$: {$I_1$,$I_2$,$I_3$} | |
| *Country* | $f_1^h$:{$I_1$,$I_2$} | | $f_1^h$:{$I_1$,$I_2$} | |

**Table 3.3.3.2: Fragments of classes – Deleting Class**

### *3.3.4 Changes in instance objects*

This section uses the OOHF algorithm to generate horizontal fragments from scratch, when new instances of objects are added into class. It uses the original database schema of figure 3.2.4 and new instances of figure 3.2.4.1 together. It generates minterm predicates and primary fragments for all classes based on the application queries of figure 3.2.7. The primary fragments of classes are

| | | | |
|---|---|---|---|
| School | $f_1^p$: {$I_1$, $I_2$, $I_3$, $I_5$, $I_8$} | $f_2^p$: {$I_4$, $I_6$, $I_7$} | |
| Hospital | $f_1^p$: {$I_1$, $I_2$, $I_5$} | $f_2^p$: {$I_3$, $I_4$, $I_6$} | |
| Highway | $f_1^p$: {$I_2$, $I_3$, $I_4$, $I_5$, $I_6$, $I_7$, $I_9$} | $f_2^p$: {$I_1$, $I_8$} | |
| City | $f_1^p$:{$I_1$, $I_4$, $I_6$} | $f_2^p$:{$I_2$, $I_3$, $I_5$, $I_7$} | |
| State | $f_1^p$:{$I_1$} | $f_2^p$:{$I_2$} | $f_2^p$:{$I_3$}. |

Then, it generates derived fragments for each member classes of the class. For example, the derived fragments of class City, based on primary fragments of class School are $f_1^d$:{$I_1$, $I_2$, $I_4$} and $f_2^d$:{$I_3$, $I_2$, $I_5$} and based on primary fragments of class Hospital are $f_3^d$:{$I_1$, $I_2$, $I_4$} and $f_4^d$:{$I_2$, $I_3$, $I_5$}. The derived fragments of class Road, based on primary fragments of class Highway are $f_1^d$:{$I_1$, $I_2$, $I_3$, $I_4$, $I_5$} and $f_2^d$:{$I_1$, $I_6$}. The derived fragments of class State, based on primary fragments of class City are $f_1^d$:{$I_1$, $I_3$} and $f_2^d$:{$I_1$, $I_2$, $I_3$}, based on the derived fragments of class City are $f_3^d$:{ $I_1$, $I_3$}, $f_4^d$: {$I_1$,$I_2$, $I_3$}, $f_5^d$: {$I_1$, $I_3$} and $f_6^d$: { $I_1$, $I_2$, $I_3$}, and based on the derived fragments of class Road are $f_7^d$: {$I_1$, $I_2$, $I_3$} and $f_8^d$: {$I_1$, $I_3$}. The derived fragments of class Country, based on primary fragments of class State are $f_1^d$:{$I_1$}, $f_2^d$:{$I_1$} and $f_3^d$:{$I_2$} and based on

derived fragments of class State are $f_4^d$:{$I_1$, $I_2$}, $f_5^d$: {$I_1$, $I_2$}, $f_6^d$: {$I_1$, $I_2$}, $f_7^d$: {$I_1$, $I_2$}, $f_8^d$:{$I_1$, $I_2$}, $f_9^d$: {$I_1$, $I_2$}, $f_{10}^d$: {$I_1$, $I_2$} and $f_{11}^d$: {$I_1$, $I_2$}.

Now, it integrates primary and derived fragments to generate the horizontal fragments. It merges derived fragments with primary fragment that has maximum affinity with. For example, for class City, $f_1^d$, $f_2^d$, $f_3^d$, and $f_4^d$ have the maximum affinity with $f_1^p$, $f_2^p$, $f_1^p$ and $f_2^p$ of 5, 10, 5 and 10 respectively. After merging these derived fragments with corresponding primary fragments, we get $f_1^h$:{$I_1$, $I_2$, $I_4$, $I_6$} and $f_2^h$:{$I_2$, $I_3$, $I_5$, $I_7$}. Here, $I_2$ are overlapping in different fragments. Applying affinity rule 2.2, $I_2$ is placed in $f_1^h$ with maximum affinity of 45. Hence, the final fragments are $f_1^h$:{$I_1$, $I_2$, $I_4$, $I_6$} and $f_2^h$:{ $I_3$, $I_5$, $I_7$}. As class Road does not have any primary fragments, so the derived fragments will be the horizontal fragments for the class. Here, $I_1$ overlaps in different fragments. Applying affinity rule 2.2 we get, $I_1$ is placed in $f_1^d$ with maximum affinity 60. Hence, the final horizontal fragments are $f_1^h$:{$I_1$, $I_2$, $I_3$, $I_4$, $I_5$} and $f_2^h$: {$I_6$}. For class State, $f_1^d$, $f_2^d$, $f_3^d$, $f_4^d$, $f_5^d$, $f_6^d$, $f_7^d$ and $f_8^d$ have maximum affinity with $f_1^p$, $f_2^p$, $f_1^p$, $f_2^p$, $f_1^p$, $f_2^p$, $f_2^p$ and $f_1^p$ of 0, 0, 0, 0, 0, 0, 0 and 0 respectively. After merging these derived fragments with corresponding primary fragments, we get $f_1^h$:{$I_1$, $I_3$} and $f_2^h$:{$I_1$, $I_2$, $I_3$}. Here, $I_1$ and $I_2$ are overlapping in different fragments. Applying affinity rule 2.2, $I_1$ and $I_2$ are placed in $f_2^h$ with maximum affinity of 25 and 30 respectively. Hence, the final horizontal fragment is $f_1^h$: {$I_1$, $I_2$, $I_3$}. As class Country does not have any primary fragments, so the derived fragments will be the horizontal fragments of the class. Here, $I_1$ and $I_2$ are overlapping in different fragments. Applying affinity rule 2.2, $I_1$ and $I_2$ are placed in $f_4^d$ with maximum affinity of 15 and 15 respectively. Hence, the final horizontal fragment is $f_1^h$: {$I_1$, $I_2$}.

The following table shows fragments generated by IOHF algorithm are identical to the fragments generated by OOHF algorithm for each class, after adding instances.

| Classes | Fragments using OOHF Algorithm | | Fragments using IOHF Algorithm | |
|---|---|---|---|---|
| *School* | $f_1^h$:{$I_1$,$I_2$,$I_3$,$I_5$,$I_8$} | $f_2^h$:{$I_4$,$I_6$,$I_7$} | $f_1^h$:{$I_1$,$I_2$,$I_3$,$I_5$,$I_8$} | $f_2^h$:{$I_4$,$I_6$,$I_7$} |
| *Hospital* | $f_1^h$:{$I_1$,$I_2$,$I_5$} | $f_2^h$:{$I_3$,$I_4$,$I_6$} | $f_1^h$:{$I_1$,$I_2$,$I_5$} | $f_2^h$:{$I_3$,$I_4$,$I_6$} |
| *Highway* | $f_1^h$:{$I_2$,$I_3$,$I_4$,$I_5$,$I_6$,$I_7$,$I_9$} | $f_2^h$:{$I_1$,$I_8$} | $f_1^h$:{$I_2$,$I_3$,$I_4$,$I_5$,$I_6$,$I_7$,$I_9$} | $f_2^h$:{$I_1$,$I_8$} |
| *City* | $f_1^h$:{$I_1$,$I_2$,$I_4$,$I_6$} | $f_2^h$:{$I_3$,$I_5$,$I_7$} | $f_1^h$:{$I_1$,$I_2$,$I_4$,$I_6$} | $f_2^h$:{$I_3$,$I_5$,$I_7$} |
| *Road* | $f_1^h$:{$I_1$,$I_2$,$I_3$,$I_4$,$I_5$} | $f_2^h$:{$I_6$} | $f_1^h$:{$I_1$,$I_2$,$I_3$,$I_4$,$I_5$} | $f_2^h$:{$I_6$} |
| *State* | $f_1^h$: {$I_1$,$I_2$,$I_3$} | | $f_1^h$: {$I_1$,$I_2$,$I_3$} | |
| *Country* | $f_1^h$:{$I_1$,$I_2$} | | $f_1^h$:{$I_1$,$I_2$} | |

**Table 3.3.4.1: Fragments of classes – Adding Instances**

After deleting existing instances $I_2$ and $I_4$ from class School, $I_3$ from class City and $I_4$ from Hospital of database schema, this section fragments object based system using OOHF algorithm. It applies the minterm predicates based on the application queries of figure 3.2.7 and generates primary fragments for all classes. The primary fragments of classes are

| | | | |
|---|---|---|---|
| School | $f_1^p$: {$I_1$, $I_3$, $I_5$} | $f_2^p$: {$I_6$} | |
| Hospital | $f_1^p$: {$I_1$, $I_2$, $I_5$} | $f_2^p$: {$I_3$, $I_6$} | |
| Highway | $f_1^p$: {$I_2$, $I_3$, $I_4$, $I_5$, $I_6$, $I_7$} | $f_2^p$: {$I_1$, $I_8$} | |
| City | $f_1^p$: {$I_1$, $I_4$} | $f_2^p$: {$I_2$, $I_5$} | |
| State | $f_1^p$: {$I_1$} | $f_2^p$:{$I_2$} | $f_2^p$: {$I_3$} |

Then, it generates derived fragments for each member classes of the class. For example, the derived fragments of class City, based on primary fragments of class Hospital are $f_1^d$: {$I_1$, $I_2$, $I_4$} and $f_2^d$: {$I_2$, $I_5$}, and based on primary fragments of class School are: $f_3^d$: {$I_1$, $I_2$, $I_4$} and $f_4^d$:{$I_5$}. The derived fragments of class Road, based on primary fragments of class Highway are $f_1^d$:{$I_1$, $I_2$, $I_3$, $I_4$, $I_5$} and $f_2^d$:{$I_1$, $I_6$}. The derived fragments of class State based on primary fragments of class City are $f_1^d$:{$I_1$, $I_3$} and $f_2^d$:{$I_1$, $I_3$}, based on the derived fragments of class City are $f_3^d$:{ $I_1$, $I_3$}, $f_4^d$:{$I_1$, $I_3$}, $f_5^d$:{$I_1$, $I_3$} and $f_6^d$:{$I_3$} and based on the derived fragments of class Road are $f_7^d$:{$I_1$, $I_2$, $I_3$} and $f_8^d$:{$I_1$, $I_3$}. The derived fragments of class Country, based on primary fragments of class State are $f_1^d$:{$I_1$}, $f_2^d$:{$I_1$} and $f_3^d$:{$I_2$} and based on derived fragments of class State are $f_4^d$:{$I_1$, $I_2$}, $f_5^d$:{$I_1$, $I_2$}, $f_6^d$:{$I_1$, $I_2$}, $f_7^d$:{$I_1$, $I_2$}, $f_8^d$:{$I_1$, $I_2$}, $f_9^d$:{$I_2$}, $f_{10}^d$:{$I_1$, $I_2$} and $f_{11}^d$:{$I_1$, $I_2$}.

Now, it integrates primary and derived fragments to generate the horizontal fragments. It merges derived fragments with primary fragment that has maximum affinity with. For example, for class City, $f_1^d$, $f_2^d$, $f_3^d$, and $f_4^d$ have the maximum affinity with $f_1^p$, $f_2^p$, $f_1^p$ and $f_2^p$ of 15, 10, 15 and 0 respectively. After merging these derived fragments with corresponding primary fragments, we get, $f_1^h$:{$I_1$, $I_2$, $I_4$} and $f_2^h$:{$I_2$, $I_5$}. Here, $I_2$ is overlapping in different fragments. Applying affinity rule 2.2, $I_2$ is placed in $f_1^h$ with maximum affinity of 30. Hence, the final fragments are $f_1^h$:{$I_1$, $I_2$, $I_4$} and $f_2^h$:{$I_5$}. As class Road does not have any primary fragments, so the derived fragments will be the horizontal fragments of the class. Here, $I_1$ overlaps in different fragments. Applying affinity rule 2.2, $I_1$ is placed in $f_1^d$ with maximum affinity 60. Hence, the final horizontal fragments are $f_1^h$:{$I_1$, $I_2$, $I_3$, $I_4$, $I_5$} and $f_2^h$: {$I_6$}. For class State, $f_1^d$, $f_2^d$, $f_3^d$, $f_4^d$, $f_5^d$, $f_6^d$, $f_7^d$ and $f_8^d$ have maximum affinity with $f_1^p$, $f_1^p$, $f_1^p$, $f_1^p$, $f_1^p$, $f_3^p$, $f_2^p$ and $f_1^p$ of 0, 0, 0, 0, 0, 5, 0 and 0 respectively. After merging these derived fragments with corresponding primary fragments, we get, $f_1^h$:{$I_1$, $I_3$} and $f_2^h$:{$I_1$, $I_2$, $I_3$}. Here, $I_1$ and $I_3$ are overlapping in different fragments. Applying affinity rule 2.2 we get, $I_1$ and $I_3$ are placed in $f_2^h$ with maximum affinity of 25 and 25 respectively. Hence, the final horizontal fragment is $f_1^h$:{$I_1$, $I_2$, $I_3$}. As class Country does not have any primary fragments, so the derived fragments will be the primary fragments of the class. Here, $I_1$ and $I_2$ are overlapping in different fragments. Applying affinity rule 2.2 we get, $I_1$ and $I_2$ are placed in $f_4^d$ with maximum affinity of 15 and 15 respectively. Hence, the final horizontal fragment is $f_1^h$: {$I_1$, $I_2$}.

The following table shows fragments generated by IOHF algorithm are identical to the fragments generated by OOHF algorithm for each class, after deleting instances.

| **Classes** | **Fragments using OOHF Algorithm** | | **Fragments using IOHF Algorithm** | |
|---|---|---|---|---|
| *School* | $f_1^h$:{$I_1$,$I_3$,$I_5$} | $f_2^h$:{$I_6$} | $f_1^h$:{$I_1$,$I_3$,$I_5$} | $f_2^h$:{$I_6$} |
| *Hospital* | $f_1^h$:{$I_1$,$I_2$,$I_5$} | $f_2^h$:{$I_3$,$I_6$} | $f_1^h$:{$I_1$,$I_2$,$I_5$} | $f_2^h$:{$I_3$,$I_6$} |
| *Highway* | $f_1^h$:{$I_2$,$I_3$,$I_4$,$I_5$,$I_6$,$I_7$} | $f_2^h$:{$I_1$,$I_8$} | $f_1^h$:{$I_2$,$I_3$,$I_4$,$I_5$,$I_6$,$I_7$} | $f_2^h$:{$I_1$,$I_8$} |
| *City* | $f_1^h$:{$I_1$,$I_2$,$I_4$} | $f_2^h$:{$I_5$} | $f_1^h$:{$I_1$,$I_2$,$I_4$} | $f_2^h$:{$I_5$} |
| *Road* | $f_1^h$:{$I_1$,$I_2$,$I_3$,$I_4$,$I_5$} | $f_2^h$:{$I_6$} | $f_1^h$:{$I_1$,$I_2$,$I_3$,$I_4$,$I_5$} | $f_2^h$:{$I_6$} |
| *State* | $f_1^h$: {$I_1$,$I_2$,$I_3$} | | $f_1^h$: {$I_1$,$I_2$,$I_3$} | |
| *Country* | $f_1^h$:{$I_1$,$I_2$} | | $f_1^h$:{$I_1$,$I_2$} | |

**Table 3.3.4.2: Fragments of classes – Deleting Instances**

# Chapter 4: Performance Analysis

In this chapter, we compare the performance of IOHF algorithm with OOHF algorithm. The IOHF algorithm is implemented as described in chapter 3. All experiments are performed on a 733 MHz computer with 128 megabytes of memory. The operating system is Windows 2000. All algorithms are written in Visual C++ language and running under Microsoft Visual Studio.

## 4.1 Theoretical Analysis

We can define the cost of horizontal fragmentation ($C_{fc}$) of a class by the following manners:

$C_{fc}$ = Cost for minterm predicates ($C_{minterm}$)
+ Cost for primary fragments ($C_{primary}$)
+ Cost for derived fragments ($C_{derived}$)

Cost for minterm predicates ($C_{minterm}$) of a class is the sum of cost to generate minterm predicates from each simple predicate. If P is the total number of predicates from all queries, those are accessing the class, then total number of minterm predicates will be $M = 2^P$. Assuming, $C_M$ (e.g., 0.2 ms) is the cost to generate a minterm predicate from a simple predicate. Then cost for minterm predicates, $C_{minterm} = M \text{ X } C_M = 2^P \text{ X } C_M$.

Cost for primary fragments ($C_{primary}$) of a class is the sum of cost to apply a minterm predicate to an instance. If n is the total number of instances of a class and $C_p$ (e.g., 0.3 ms) is the cost to apply a minterm to an instance, then cost for primary fragments, $C_{primary} = n \text{ X } M \text{ X } C_p = n \text{ X } 2^P \text{ X } C_p$.

Cost for derived fragments ($C_{derived}$) of a class is the sum of cost for each subclass to select the parent pointer for each instance of each primary fragment. Assuming, $C_d$ (e.g., 0.4 ms) is the cost to find the parent pointer, $n_p$ is the number of instances of a primary fragment, $F_p$ is the total number of primary fragments of a subclass and $S_n$ is the number of subclasses, then cost for derived fragments, $C_{derived} = \sum_{i=0}^{S_n} \sum_{j=1}^{F_p} (n_p \text{ X } C_d)$.

The total horizontal fragmentation cost of an object based systems is the sum of fragmentation cost of each class. If there are N nodes (classes) in the class inheritance hierarchy, then the total horizontal fragmentation cost is $C_T = \sum_{i=1}^{N} C_{fc}$.

The advantage of IOHF algorithm, over OOHF algorithm is that, it avoids computing for some nodes (classes) of class inheritance hierarchy of object bases systems. For example, from section 3.2.1, when we added new queries (figure 3.2.1.1) in class School, class City and class Road of class inheritance hierarchy (figure 3.2.5), IOHF algorithm generates minterm predicates based on these new simple predicates for these classes. Then, it generates primary fragments of these classes and derived fragments of class City, class State and class Country. So, the total cost of refragmentation of the object based system will be,

$$C_T = C_{fc(school)} + C_{fc(City)} + C_{fc(Road)} + C_{fc(State)} + C_{fc(Country)}$$

Here, IOHF calculates, the cost of class fragmentation, based on new predicates. For example, there are two predicates $P_1$ and $P_2$ in class School for new query. Again, as class State and class Country do not have any new query, it applies only $C_{derived}$ cost for these classes.

On the other hand, OOHF algorithm refragments all classes based on existing and new queries. For example, class School has four predicates, $P_1$, $P_2$, $P_3$ and $P_4$. So, the total cost of refragmentation of the class inheritance hierarchy will be,

$$C_T = C_{fc(School)} + C_{fc(Hospital)} + C_{fc(City)} + C_{fc(Highway)} + C_{fc(Road)} + C_{fc(State)} + C_{fc(Country)}$$

Similarly, IOHF algorithm avoids computing, for deleting queries, changes in access frequencies, adding classes, deleting classes, adding instances and deleting instances.

The disadvantage of IOHF algorithm over OOHF algorithm is that, when the last query is deleted from a class, IOHF algorithm generates minterm predicates, primary and derived fragments for this class. On the other hand, instead of applying any query, OOHF algorithm assumes all instances of the class are in the same fragments. For example, from section 3.2.1, when we deleted the query 2 (figure 3.2.1.1), IOHF algorithm generates minterm predicates and primary fragments of class School and derived fragment of class City, class State and class Country. OOHF algorithm does not apply any query for class School. Though OOHF algorithm gains the $C_{fc(School)}$ for class School, but total fragmentation cost ($C_T$) also depends on the $C_{fc}$ of

other classes. As a result, after deleting queries in section 3.2.1, IOHF algorithm has better performance than OOHF algorithm. From OOHF algorithm, we can get better performance, only when we delete all queries from an object based system.

# 4.2 Experimental Evaluation

Our synthetic object based datasets are generated using a customize program, for class inheritance hierarchies of figure 3.2.5 and figure 3.2.3.1. The data of each class consists instances, where an instance includes attributes, methods and pointer to the parent class. Three different sets of data with different number of instances for each class are used as input to the algorithms to compare the performance between IOHF and OOHF algorithms. The following sections show the performance results.

### *4.2.1 Execution Time for Dataset 1*

Table 4.2.1.1 shows number of instances and the changes in inputs for each class to compare the performance of IOHF and OOHF. Here, column titles, AQ means adding queries, DQ means deleting queries, AF means changes in access frequencies, AC means adding classes, DC means deleting classes, AI means adding instances and DI means deleting instances. For example, data of column AQ indicates, one query is added into class City, one query is added into class School and one query is added into class Road. The data of column AQ for class Zone represents, one query is added into the class Zone, when class Zone is added into the class inheritance hierarchy.

| Class Name | Number of Instances | Changes in Inputs | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | AQ | DQ | AF | AC | DC | AI | DI |
| Country | 2 | | | | | | | |
| State | 3 | | | | | | | |
| City | 5 | 1 | 1 | 1 | | | 2 | 1 |
| School | 6 | 1 | 1 | | | 1 | 2 | 2 |
| Hospital | 5 | | | | | | | 1 |
| Road | 6 | 1 | | | | | | |
| Highway | 8 | | | | | | 1 | |
| Urbanroad | 6 | | | | | | | |
| Zone | 7 | 1 | | | 1 | | | |

**Table 4.2.1.1: Number of Instances and Changes in Inputs of Classes – Dataset 1**

Table 4.2.1.2 shows the execution time for each type of changes in input using OOHF and IOHF algorithms. Figure 4.2.1.1 shows the bar graph based on the execution time from table 4.2.1.2 for each class.

| Changes in Inputs | Execution Time (ms) - OOHF | Execution Time (ms) – IOHF | Time Gain (ms) |
|---|---|---|---|
| Adding Queries (AQ) | 110.679 | 80.279 | 30.400 |
| Deleting Queries (DQ) | 78.657 | 60.742 | 17.916 |
| Access Frequencies (AF) | 94.700 | 43.935 | 50.765 |
| Adding Classes (AC) | 170.249 | 135.486 | 34.763 |
| Deleting Classes (DC) | 79.284 | 34.286 | 45.687 |
| Adding Instances (AI) | 94.284 | 65.461 | 28.823 |
| Deleting Instances (DI) | 89.120 | 60.628 | 28.492 |

**Table 4.2.1.2: Execution Time (ms) – Dataset 1**

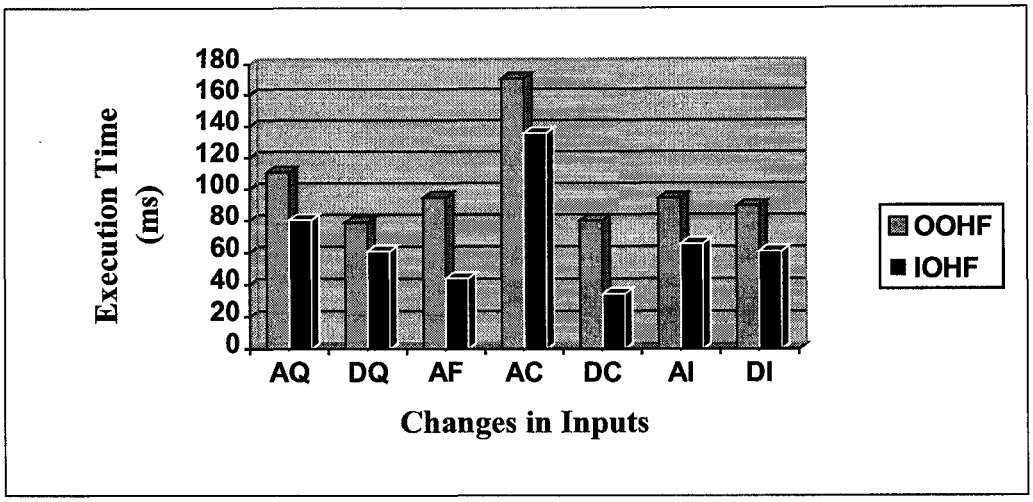


**Figure 4.2.1.1: Execution Time (Bar Graph) – Dataset 1**

From table 4.2.1.2 and figure 4.2.1.1, we can find that IOHF algorithm requires less time than OOHF algorithm to refragment the object based system.

### *4.2.2 Execution Time for Dataset 2*

Table 4.2.2.1 shows number of instances and the changes in inputs for each class to compare the performance of IOHF and OOHF.

| Class Name | Number of Instances | Changes in Inputs | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | AQ | DQ | AF | AC | DC | AI | DI |
| Country | 2 | | | | | | | |
| State | 16 | | | | | | | |
| City | 44 | 1 | 1 | 1 | | | 3 | 1 |
| School | 78 | 1 | 1 | | | 1 | 3 | 2 |
| Hospital | 46 | | | | | | | 1 |
| Road | 45 | 1 | | | | | | |
| Highway | 80 | | | | | | 1 | |
| Urbanroad | 6 | | | | | | | |
| Zone | 172 | 1 | | | 1 | | | |

**Table 4.2.2.1: Number of Instances and Changes in Inputs of Classes – Dataset 2**

Table 4.2.2.2 shows the execution time for each type of changes in input using OOHF and IOHF algorithms. Figure 4.2.2.1 shows the bar graph based on the execution time from table 4.2.2.2 for each class.

| Changes in Inputs | Execution Time (ms) – OOHF | Execution Time (ms) – IOHF | Time Gain (ms) |
|---|---|---|---|
| Adding Queries (AQ) | 3717.237 | 1336.393 | 2380.847 |
| Deleting Queries (DQ) | 2136.270 | 1868.847 | 267.423 |
| Access Frequencies (AF) | 3488.032 | 751.611 | 2736.421 |
| Adding Classes (AC) | 4026.390 | 3492.981 | 583.409 |
| Deleting Classes (DC) | 2431.120 | 865.683 | 1565.437 |
| Adding Instances (AI) | 5399.852 | 2878.061 | 2524.791 |
| Deleting Instances (DI) | 5243.790 | 2740.005 | 2503.785 |

**Table 4.2.2.2: Execution Time (ms) – Dataset 2**

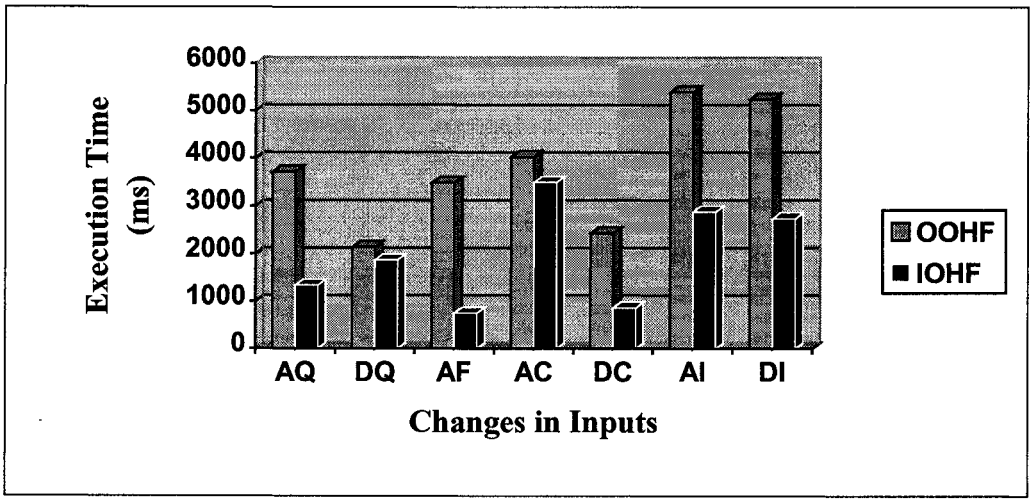


**Figure 4.2.2.1: Execution Time (Bar Graph) – Dataset 2**

From table 4.2.2.2 and figure 4.2.2.1, we can find that IOHF algorithm requires less time than OOHF algorithm to refragment the object based system.

### *4.2.3 Execution Time for Dataset 3*

Table 4.2.3.1 shows number of instances and the changes in inputs for each class to compare the performance of IOHF and OOHF.

| Class Name | Number of Instances | Changes in Inputs | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | AQ | DQ | AF | AC | DC | AI | DI |
| Country | 2 | | | | | | | |
| State | 26 | | | | | | | |
| City | 72 | 1 | 1 | 1 | | | 3 | 1 |
| School | 126 | 1 | 1 | | | 1 | 3 | 2 |
| Hospital | 90 | | | | | | | 1 |
| Road | 73 | 1 | | | | | | |
| Highway | 128 | | | | | | 1 | |
| Urbanroad | 6 | | | | | | | |
| Zone | 288 | 1 | | | 1 | | | |

**Table 4.2.3.1: Number of Instances and Changes in Inputs of Classes – Dataset 3**

Table 4.2.3.2 shows the execution time for each type of changes in input using OOHF and IOHF algorithms. Figure 4.2.3.1 shows the bar graph based on the execution time from table 4.2.3.2 for each class.

| Changes in Inputs | Execution Time (ms) – OOHF | Execution Time (ms) – IOHF | Time Gain (ms) |
|---|---|---|---|
| Adding Classes (AQ) | 9509.668 | 2602.706 | 6906.962 |
| Deleting Classes (DQ) | 6005.153 | 4223.312 | 1781.842 |
| Access Frequencies (AF) | 9224.916 | 1738.667 | 7486.249 |
| Adding Classes (AC) | 10130.631 | 8947.833 | 1182.798 |
| Deleting Classes (DC) | 6630.042 | 1962.656 | 4467.385 |
| Adding Instances (AI) | 9524.735 | 7880.483 | 1644.251 |
| Deleting Instances (DI) | 9283.269 | 7725.666 | 1557.603 |

**Table 4.2.3.2: Execution Time (ms) – Dataset 3**

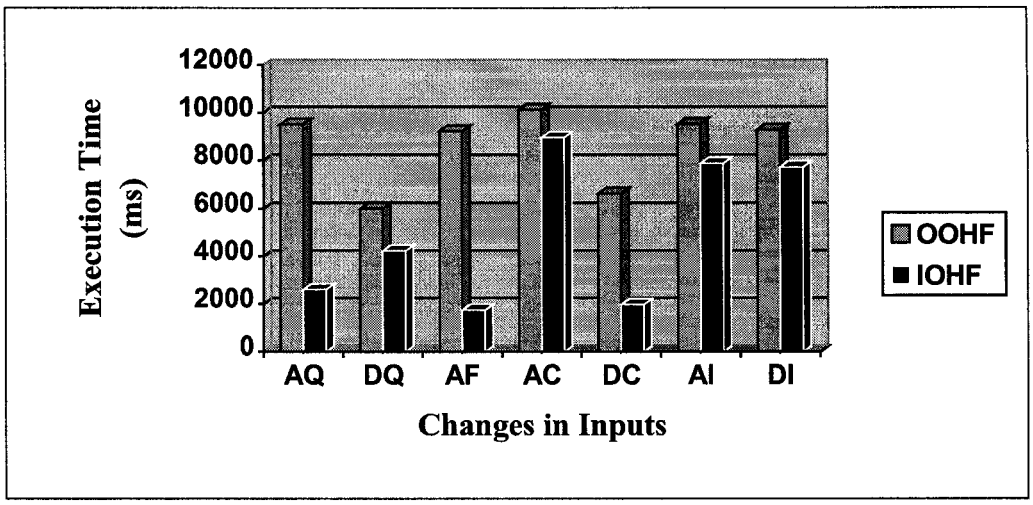


**Figure 4.2.3.1: Execution Time (Bar Graph) – Dataset 3**

From table 4.2.3.2 and figure 4.2.3.1, we can find that IOHF algorithm requires less time than OOHF algorithm to refragment the object based system [ED02].

# Chapter 5: Conclusions and Future Research

The problem of fragmenting and distributing data objects in an object-oriented database system is complex owing to features of encapsulation, inheritance and class composition relationships that need to be captured. This thesis handles a further important extension of incremental horizontal fragmentation of classes by specifying the sequence of operations to perform on existing fragments when an input data is added, deleted or changed. This algorithm consists of several sequences of operations. Generally, each sequence uses the changes including addition, deletion and moving, to define new fragments of classes, which are then merged with existing fragments of the class based on affinity rules.

This algorithm generates fragments based on changes in a particular class and then propagates only these changes to all of the member classes. Fragments of all other classes remain the same. Therefore, the re-fragmentation time for the system is reduced. For example, from section 3.2.1 and section 3.3.1, adding new application, we can find that, the OOHF has started from scratch to re-fragments all of the classes of the class inheritance hierarchy whether it has any effect due to the new application or not. On the other hand, in IOHF, we can find that, only class School, City, Road, State and Country are re-fragmented based on only new application. When re-fragmenting these classes, it generates minterm predicates and primary fragments only for new application. As a result, IOHF algorithm needs less iteration than OOHF to complete the re-fragmentation process of an object based systems. Hence, it increases the performance of a system by using a few resources.

The algorithm presented here, contributes to solve the problem of re-fragmentation of object-oriented databases system efficiently.

## 5.1 Future Research

The following aspects can be considered as future work.

1. Incremental Vertical and Hybrid fragmentation of object-oriented database.
2. Incremental Horizontal, Vertical and Hybrid fragmentation of relational database.

# References


[ABDDMZ89] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier and S. Zdonik, *"The Object-Oriented Database System Manifesto"*, In Proceedings of the First International Conference on Deductive and Object-Oriented Databases, pp 223-40, Kyoto, Japan, December 1989.

[BF95a] Elisa Bertino, Paola Foscoli, *"On Modeling Cost Functions for Object-Oriented Databases"*, IEEE Transcation Knowledge and Data Engineering, Vol 9. No. 3 pp 500-512, April 1995

[BKS97] Ladjel Bellatreche, Kamalakar Karlapalem, Ana Simonet, *"Horizontal Class Partioning in Object-Oriented Databases"*, DEXA '97 conference, 1997

[CM94] Soon M. Chung, Pyeong S. Mah, *"Schema integration for multidatabases using the unified relational and object-oriented model"*, Proceedings of the 1995 ACM 23rd Annual computer science conference on the shrinking footprint and growing impact, pp 208-215, CIKM 1994

[CMVN93] S. Chakravarthy, J. Muthuraj, R. Varadarajan, S. Navathe, *"An Objective Function For Vertically Partitioning Relations in Distributed Databases and its Analysis"*, Distributed and parallel Databases, 2(1):183-207, 1993

[CNP82] S. Ceri, M. Negri, G. Pelagatti, *"Horizontal data partitioning in database design"*, In proceedings of ACM SIGMOD International Conference on Management of Data , 1982

[EB95] C. I. Ezeife, K. Barker, *"A Comprehensive Approach to Horizontal Class Fragmentation in a Distributed Object Based System"* International Journal of Distributed and Parallel Database, Kluwer Academic Publisher, V1, 1995

[EB98] C. I. Ezeife, K. Barker, *"Distributed Object Based Design: Vertical Fragmentation of Classes"*, International Journal of Distributed and Parallel Databases, Vol. 6, No. 4, pp. 327-360, Kluwer Academic Publishers, October 1998

[ED02] C. I. Ezeife, Pinakpani Dey, *"Incremental Horizontal Fragmentation of Database Class Objects"*, submitted to 5th International Conference on Enterprise Information Systems (ICEIS 2003), April 2003.

[EN00] Ramez Elmasri, Shamkant B. Navathe, "*Fundamentals of Database Systems*", Third Edition, Addison-Wesley, 2000

[EZ99a] C. I. Ezeife, Jian Zheng, *"Dynamic Database Object Horizontal Fragmentation"*, Proceedings of the 8th International Conference on Information Systems Development-Methods and Tools, Theory and Practice, Boise, Plennum Press Publishers, August 1999

[EZ99b] C. I. Ezeife, Jian Zheng, *"Measuring the Performance of Database Object Horizontal Fragmentation Schemes"*, Proceedings of the 3rd International Database Engineering and Applications Symposium (IDEAS 99), IEEE Publication, Montreal, August 1999

[FEB98] Philippe Futtersack, Christophe Espert, Didier Bolf, *"The Electronic Library Project: SGML Document Management System Based on ODBMS"*, SIGMOD Record, Vol. 27, No. 1, March 1998

[GA98] Jeff Garland, Dick Anthony, *"Using Objectivity on the IRIDIUM System"*, SIGMOD Record, Vol. 27, No. 1, March 1998

[Ha98] P. G. Hardy, *"Map Production from an active Object Database using Dynamic Representation and Automated Generalisation"*, Laser-Scan Ltd, Science Park, Milton Road, Cambridge, CB4 0FY, UK. http://oldsite.laser-scan.com/papers/index.htm

[HAG98] David Hansen, Daniel Adams, Deborah Gracio, *"In the Trenches with ObjectStore"*, SIGMOD Record, Vol. 27, No. 1, March 1998

[KG94] Alfons Kemper, Guido Moerkotte, *"Object-Oriented Database Management: Applications in Engineering and Computer Science"*, Prentice Hall, 1994

[KIM90] Won Kim, *"Introduction to Object-Oriented Databases"*, The MIT Press, Cambridge, Massachusetts, London, England, 1990

[KL95] K. Karlapalem, Q. Li, "*Partitioning Schemes for Object Oriented Databases*", 5th International Workshop on Research Issues on Data Engineering: Distributed Object Management (RIDE-DOM'95), pages 42-49, 1995

[KMN93] K. Karlapalem, M. M. A. Morsi, S. B. Navathe, *"Issues in Distribution Design of Object-Oriented Databases"*, Distributed Object Management, (edited by T. Ozsu, U. Dayal, and P. Valduriez), pages 148-165, Morgan Kauffman, 1993

[LN96] S. J. Lim, Y. K. Ng., *"A formal approach for horizontal fragmentation in distributed deductive database design"*, 7th international conferences on Database and Expert Systems Applications, 1996

[MK95] Savonnet M., Yetongnon K., *"A qualitative approach for class fragmentation in distributed object oriented databases"*, Proceedings of the 5th Annual Workshop on Information Technologies and Systems. , WIT'S 1995

[OV99] M. Tamer Ozsu, Patric Valduriez, Prentice Hall, *"Principle of Distributed Database Systems"*, 1999

[Ram98] Raghu Ramakrishnan, *"Database Management Systems"*, Chapter 21, Object-Database Systems, Page 614-644

[Rao94] Bindu R. Rao, *"Object-Oriented Databases Technology Applications and Products"*, McGraw-Hill, Inc. 1994

[RBKW91] Fausto Rabitti, Elisa Bertino, Won Kim, Darrell Woelk, *"A model of authorization for next-generation database systems"*, ACM Transaction on Database Systems, Vol. 16, No. 1, pp 88-131, March 1991

[RPMP99] Ivan Radev, Niki Pissinou, Kia Makki, E. K. Park, *"Graph-Based Object-Oriented Approach for Structural and Behavioral Representation of Multimedia Data"*, Eight International Conference on Information Knowledge Management (CIKM'99), November 1999

[Ru94] Elke Angelika Rundensteiner, *"A classification algorithm for supporting object-oriented views"*, Proceedings of the third international conference on Information and knowledge management, pp 18-25, CIKM 1994

[RZ95] Franck Ravat, Gilles Zurfluh, *"Issues in the Fragmentation of Object Oriented Database"*, Proceedings. Basque International Workshop on Information Technology, 1995

[SM96] Michael Stonebraker, Dorothy Moore, *"Object-Relational DBMSs: The Next Great Wave"*, Morgan Kaufmann Publishers, Inc., 1996

[Sz96] Ivan Szanto, *"Extended Relational Database"*, http://tu.dhs.org/oodbs/tit-node4.html, 1996

[Th01] Dr. Grant Thrall, *"Lecture 3 – 29th January: Spatial Data, Data Models, and Modeling the Real World"*, Department of Geography, University of Florida, 2001, http://www.clas.ufl.edu/users/mbinford/geo3171/LectureNotes/Lecture_3_spatial_data.pdf

[WT96] Stephen Wong, Satoshi Tojo, *"A Deductive Object-Oriented Database System for Situated Inference in Law"*, IEEE Transactions on Knowledge and Data Engineering, Vol 8. No3., June 1996

# Vita Auctoris

| | |
|---|---|
| Name | Pinakpani Dey |
| Year of birth | 1970 |
| Place of birth | Sylhet, Bangladesh |
| Education | M. Sc., Computer Science<br>University of Windsor<br>Windsor, Ontario<br>Canada<br>2000-2002 |
| | B. Sc., Computer Science and Engineering<br>Bangladesh University of Engineering and Technology (BUET)<br>Dhaka<br>Bangladesh<br>1988-1992 |