1/21/2020

ENTERPRISE DATABASE ADMINISTRATION ASSIGNMENTS – INDIVIDUAL

Name: Henok Gelaneh

ID: ATR/7217/10
Department: ITSC
Section: IT
Subject: Database Administration

Table of Contents

Relational Algebra and Relational Calculus

Introduction

A **Relational Database management System**(RDBMS) is a database management system based on the relational model introduced by E.F Codd. In relational model, data is stored in **relations**(tables) and is represented in form of **tuples**(rows). **RDBMS** is used to manage Relational database. **Relational database** is a collection of organized set of tables related to each other, and from which data can be accessed easily. Relational Database is the most commonly used database these days.

- **Table** In Relational database model, a **table** is a collection of data elements organised in terms of rows and columns. A table is also considered as a convenient representation of **relations**. But a table can have duplicate row of data while a true **relation** cannot have duplicate data. Table is the most simplest form of data storage.
- **Tuple** A single entry in a table is called a **Tuple** or **Record** or **Row**. A **tuple** in a table represents a set of related data.
- **Attribute** A table consists of several records(row), each record can be broken down into several smaller parts of data known as **Attributes**. The above **Employee** table consist of four attributes, **ID**, **Name**, **Age** and **Salary**.
- **Attribute Domain** When an attribute is defined in a relation(table), it is defined to hold only a certain type of values, which is known as **Attribute Domain**.
- **Relation Schema** A relation schema describes the structure of the relation, with the name of the relation(name of table), its attributes and their names and type.
- **Relation Key** A relation key is an attribute which can uniquely identify a particular tuple(row) in a relation(table).

Relational Integity Constraints

Every relation in a relational database model should abide by or follow a few constraints to be a valid relation, these constraints are called as **Relational Integrity Constraints**.

The three main Integrity Constraints are:

1. Key Constraints
2. Domain Constraints
3. Referential integrity Constraints

- **Key Constraints** We store data in tables, to later access it whenever required. In every table one or more than one attributes together are used to fetch data from tables. The **Key Constraint** specifies that there should be such an attribute(column) in a relation(table), which can be used to fetch data for any tuple(row).The Key attribute should never be **NULL** or same for two different row of data.
- **Domain Constraint** Domain constraints refers to the rules defined for the values that can be stored for a certain attribute. We cannot store **Address** of employee in the column for **Name**. Similarly, a mobile number cannot exceed 10 digits.
- **Referential Integrity Constraint** If a table reference to some data from another table, then that table and that data should be present for referential integrity constraint to hold true.

Procedural Query language: In procedural query language, user instructs the system to perform a series of operations to produce the desired results. Here users tells what data to be retrieved from database and how to retrieve it.

Non-procedural query language: In Non-procedural query language, user instructs the system to produce the desired result without telling the step by step process. Here users tells what data to be retrieved from database but doesn't tell how to retrieve it.

Relational Algebra

Every database management system must define a query language to allow users to access the data stored in the database. **Relational Algebra** is a procedural query language used to query the database tables to access data in different ways.

In relational algebra, input is a relation(table from which data has to be accessed) and output is also a relation(a temporary table holding the data asked for by the user).

Relational Algebra works on the whole table at once, so we do not have to use loops etc to iterate over all the rows(tuples) of data one by one. All we have to do is specify the table name from which we need the data, and in a single line of command, relational algebra will traverse the entire given table to fetch data for you.

Basic/Fundamental Operations:

1. Select (σ)
2. Project (∏)
3. Union (∪)
4. Set Difference (-)
5. Cartesian product (X)
6. Rename (ρ)

Derived Operations:

1. Natural Join (⋈)
2. Left, Right, Full outer join (⟕, ⟖, ⟗)
3. Intersection (∩)
4. Division (÷)

Reference Employee table:

ID	NAME	AGE	SALARY
1	Adam	34	13000
2	Alex	28	15000
3	Stuart	20	18000
4	Ross	42	19020

Select Operation (σ)

This is used to fetch rows(tuples) from table(relation) which satisfies a given condition.

Syntax: $\sigma_p(r)$

Where, σ represents the Select Predicate, r is the name of relation(table name in which you want to look for data), and p is the prepositional logic, where we specify the conditions that must be satisfied by the data. In prepositional logic, one can use **unary** and **binary** operators like =, <, > etc, to specify the conditions.

Let's take an example of the Student table we specified above in the Introduction of relational algebra, and fetch data for **students** with **age** more than 17.

$\sigma_{age > 17}$ (Student)

This will fetch the tuples(rows) from table **Student**, for which **age** will be greater than **17**.

You can also use, and, or etc operators, to specify two conditions, for example,

$\sigma_{age > 17 \text{ and } gender = 'Male'}$ (Student)

This will return tuples(rows) from table **Student** with information of male students, of age more than 17.(Consider the Student table has an attribute Gender too.)

Project Operation (∏)

Project operation is used to project only a certain set of attributes of a relation. In simple words, If you want to see only the **names** all of the students in the **Student** table, then you can use Project Operation.

It will only project or show the columns or attributes asked for, and will also remove duplicate data from the columns.

Syntax: $\prod_{A1, A2...}(r)$

where A1, A2 etc are attribute names(column names).

For example,

$\prod_{Name, Age}(Student)$

Above statement will show us only the **Name** and **Age** columns for all the rows of data in **Student** table.

Union Operation (∪)

This operation is used to fetch data from two relations(tables) or temporary relation(result of another operation).

For this operation to work, the relations(tables) specified should have same number of attributes(columns) and same attribute domain. Also the duplicate tuples are autamatically eliminated from the result.

Syntax: A ∪ B

where A and B are relations.

For example, if we have two tables **RegularClass** and **ExtraClass**, both have a column **student** to save name of student, then,

$\prod_{Student}(RegularClass) \cup \prod_{Student}(ExtraClass)$

Above operation will give us name of **Students** who are attending both regular classes and extra classes, eliminating repetition.

Set Difference (-)

This operation is used to find data present in one relation and not present in the second relation. This operation is also applicable on two relations, just like Union operation.

Syntax: A - B

where A and B are relations.

For example, if we want to find name of students who attend the regular class but not the extra class, then, we can use the below operation:

$\prod_{Student}(RegularClass) - \prod_{Student}(ExtraClass)$

Cartesian Product (X)

This is used to combine data from two different relations(tables) into one and fetch data from the combined relation.

Syntax: A X B

For example, if we want to find the information for Regular Class and Extra Class which are conducted during morning, then, we can use the following operation:

$\sigma_{time = 'morning'}$ (RegularClass X ExtraClass)

For the above query to work, both **RegularClass** and **ExtraClass** should have the attribute **time**.

Rename Operation (ρ)

This operation is used to rename the output relation for any query operation which returns result like Select, Project etc. Or to simply rename a relation(table)

Syntax: ρ(RelationNew, RelationOld)

Let us consider two tables named as **A** and **B** **Table B:**

Table A:

ROLLNO	NAME	MARKS
1	Henok	98
3	Dawit	79
4	Sara	88

ROLLNO	NAME	MARKS
1	Henok	98
2	Meron	87
3	Dawit	79
4	Sara	88

Intersection Operator (∩)

Intersection operator is denoted by ∩ symbol and it is used to select common rows (tuples) from two tables (relations).

Let's say we have two relations R1 and R2 both have same columns and we want to select all those tuples(rows) that are present in both the relations, then in that case we can apply intersection operation on these two relations R1 ∩ R2.

Intersection works on the relation as '**this and that**'. In relational algebra, **A** ∩ **B** returns a relation instance that contains every tuple that occurs in relation to instance **A** and relation instance **B** (both together). Here, **A** and **B** need to be union-compatible, and the schema of both result and **A** must be identical.

Syntax:

```
SELECT * FROM A INTERSECT SELECT * FROM B;
```

Divide

Divide operator is used for the queries that contain the keyword ALL.

For e.g. – Find all the students who has chosen additional subjects Machine Learning and Data Mining.

Join

Join operation is as its name suggest, to join or combine two or more relations' information. Join can also be defined as a cross-product followed by selection and projection. There are several varieties of Join operation. Let's discuss all of them one by one.

Condition Join

When you want to join two relations based on the given condition, it is termed as Condition Join. It is denoted by the symbol $\bowtie_c$.

For e.g. – Select the students from Student1 table whose RollNo is greater than the RollNo of Student2 table.

Student1$\bowtie_{cStudent1.RollNo>Student2.RollNo}$Student2

Syntax:

```
SELECT * FROM Student1, Student2 WHERE Student1.RollNo > Student2.RollNo;
```

b. Equi Join

It is a special case of Condition Join. When you want to join two relations based on the equality condition, it is termed as Equi Join. It is denoted by the symbol ⋈.

For e.g. - Select the students from Student1 table whose RollNo is equalto the RollNo of Student2 table.

Student1$\bowtie_{Student1.RollNo=Student2.RollNo}$Student2

Syntax:

```
SELECT * FROM Student1, Student2 WHERE Student1.RollNo=Student2.RollNo;
```

c. Natural Join

Natural Join is that type of join in which equijoin is by default applied to all the attributes in two or more relation. Its specialty is if you want to consider the equality between two relations, you don't need to define the equality; it is predefined for all the attributes if you use Natural Join. It is denoted by the symbol ⋈.

For e.g. - Select the students from Student1 table whose RollNo is equal to the RollNo of Student2 table.

Student1⋈Student2

Syntax:

```
SELECT * FROM Student1 NATURAL JOIN Student2;
```

Relational Calculus

Contrary to Relational Algebra which is a procedural query language to fetch data and which also explains how it is done, **Relational Calculus** in non-procedural query language and has no description about how the query will work or the data will b fetched. It only focusses on what to do, and not on how to do it.

Relational Calculus exists in two forms:

1. Tuple Relational Calculus (TRC)
2. Domain Relational Calculus (DRC)

Tuple Relational Calculus (TRC)

In tuple relational calculus, we work on filtering tuples based on the given condition.

Syntax: { T | Condition }

In this form of relational calculus, we define a tuple variable, specify the table(relation) name in which the tuple is to be searched for, along with a condition.We can also specify column name using a . dot operator, with the tuple variable to only get a certain attribute(column) in result. A tuple variable is nothing but a name, can be anything, generally we use a single alphabet for this, so let's say T is a tuple variable.

To specify the name of the relation(table) in which we want to look for data, we do the following: Relation(T), where T is our tuple variable.

For example if our table is **Student**, we would put it as Student(T)

Then comes the condition part, to specify a condition applicable for a particluar attribute(column), we can use the . dot variable with the tuple variable to specify it, like in table **Student**, if we want to get data for students with age greater than 17, then, we can write it as,

T.age > 17, where T is our tuple variable.

Putting it all together, if we want to use Tuple Relational Calculus to fetch names of students, from table **Student**, with age greater than **17**, then, for T being our tuple variable,

```
T.name | Student(T) AND T.age > 17
```

Domain Relational Calculus (DRC)

In domain relational calculus, filtering is done based on the domain of the attributes and not based on the tuple values.

Syntax: { c1, c2, c3, ..., cn | F(c1, c2, c3, ... ,cn)}

where, c1, c2... etc represents domain of attributes(columns) and F defines the formula including the condition for fetching the data.

For example,

{< name, age > | ∈ Student ∧ age > 17}

Again, the above query will return the names and ages of the students in the table **Student** who are older than 17.

Difference between Relational Algebra and Relational Calculus:

	Relational Algebra	*Relational Calculus*
1.	It is a Procedural language.	While Relational Calculus is Declarative language.
2.	Relational Algebra means how to obtain the result.	While Relational Calculus means what result we have to obtain.
3.	In Relational Algebra, The order is specified in which the operations have to be performed.	While in Relational Calculus, The order is not specified.
4.	Relational Algebra is independent on domain.	While Relation Calculus can be a domain dependent.
5.	Relational Algebra is nearer to a programming language.	While Relational Calculus is not nearer to programming language.

Disjunctive and Conjunctive Normal Forms

Every truth table (Boolean function) can be written as either a conjunctive normal form (CNF) or disjunctive normal form (DNF)

A Boolean expression is in disjunctive normal form (DNF) if:

1. the variables within each term are **AND**ed together,
2. the terms are **OR**ed together, and
3. every variable or its complement is represented in every term (i.e. either A or ~A is in each term, B or ~B is in each term, etc.).
4. No parentheses or other Boolean operations appear in the expression.

A propositional formula is in disjunctive normal form (DNF) if it is the disjunction of conjunctions of literals.

- A literal is a Boolean variable, or the negation of a Boolean variable (e.g., P,¬P).
- A conjunction is a logical formula that is the AND (∧) of (smaller) formulas (e.g., P∧¬Q∧R).
- A disjunction is a logical formula that is the OR (∨) of (smaller) formulas (e.g., P∨¬Q∨R).

With this in mind, the meaning of DNF should be clearer: a DNF formula is an OR of AND's. For example,

$$X = (A \wedge \neg B \wedge \neg D) \vee (B \wedge C) \vee (C \wedge \neg D \wedge E)$$

is a DNF formula: the literals of X are A,¬B,¬D,B,C,¬D, and E; the conjunctions are A∧¬B∧ ¬D,B∧C, and C∧¬D∧E; and X is the disjunction (OR) of these three conjunctions. On the other hand,

$$Y = (A \vee \neg B) \wedge (B \vee C \vee \neg D)$$

is not a DNF formula, because the structure of the operators is inverted. But this formula is in the other "standard" form (CNF) for propositional formulas.

The rules for conjunctive normal form (CNF) are similar to the rules for DNF except for the precedence of the ANDs and ORs. A Boolean expression is in CNF if:

1. the variables within each term are **OR**ed together,
2. the terms are **AND**ed together, and
3. every variable or its complement is represented in every term (i.e. either A or ~A is in each term, B or ~B is in each term, etc.).
4. No parenthesis - other than those separating the terms - or other Boolean operations appear in the expression.

A propositional formula is in conjunctive normal form (CNF) if it is the conjunction of disjunctions of literals.

As noted above Y is a CNF formula because it is an AND of OR's. The literals of Y are A,¬B, B, C, and ¬D; the disjunctions are (A∨¬B) and (B∨C∨¬D); and Y is the conjunction (AND) of the two disjunctions.

Any propositional formula is tautologically equivalent to some formula in disjunctive and conjunctive normal form.

Converting to DNF

We now describe a procedure that converts any propositional formula *X* into disjunctive normal form. Note that the resulting DNF is not necessarily the shortest expression equivalent to *X* (or even the shortest DNF equivalent to *X*), but the result is guaranteed to be a DNF formula. We illustrate the procedure on the following example:

$$X = \neg\Big((A \vee B) \wedge (A \vee C)\Big)$$

The first step is to write the truth table of X:

A	B	C	X
T	**T**	**T**	**F**
T	**T**	**F**	**F**
T	**F**	**T**	**F**
T	**F**	**F**	**F**
F	**T**	**T**	**F**
F	**T**	**F**	**T**
F	**F**	**T**	**T**
F	**F**	**F**	**T**

(Note that if A = **T**, then the ∧ "succeeds", so X = **F**. Otherwise, the ∧ "succeeds" only if $B = C$ = **T**, and so X = **F** in this case and X = **T** in all remaining rows.)

The second step is to identify the rows that contain X = **T**; in our case, these are rows 6, 7, and 8. Finally, we encode each row as a conjunction of the corresponding literals, and the final result is the disjunction of these conjunctions. Intuitively, we can think of the DNF we're constructing as the following expression:

(row 6)∨(row 7)∨(row 8).

In our example, we have the following encoding:

row 6 : (¬*A* ∧ *B* ∧¬*C*)

row 7 : (¬*A* ∧¬*B* ∧ *C*)

row 8 : (¬*A* ∧¬*B* ∧¬*C*).

Taking the disjunction of these conjunctions yields the following DNF formula for X:

$$(\neg A \wedge B \wedge \neg C) \vee (\neg A \wedge \neg B \wedge C) \vee (\neg A \wedge \neg B \wedge \neg C).$$

Converting to CNF

Similarly, we can convert any propositional formula into conjunctive normal form. We will illustrate the process on the same propositional formula X from the previous subsection. In this case, the intuition is the following: X = T if the truth assignment does not belong to any of the rows that evaluate to F.

NOT row 1 : ¬(*A* ∧ *B* ∧ *C*) = ¬*A* ∨¬*B* ∨¬*C*

NOT row 2 : ¬(*A* ∧ *B* ∧¬*C*) = ¬*A* ∨¬*B* ∨ *C*

NOT row 3 : ¬(*A* ∧¬*B* ∧ *C*) = ¬*A* ∨ *B* ∨¬*C*

NOT row 4 : ¬(*A* ∧¬*B* ∧¬*C*) = ¬*A* ∨ *B* ∨ *C*

NOT row 5 : ¬(¬*A* ∧ *B* ∧ *C*) = *A* ∨¬*B* ∨¬*C*.

Notice that we've applied De Morgan's Law and the double negation rule to produce a disjunction.

The final CNF is the conjunction of these disjunctions:

$$(\neg A \vee \neg B \vee \neg C) \wedge (\neg A \vee \neg B \vee C) \wedge (\neg A \vee B \vee \neg C) \wedge (\neg A \vee B \vee C) \wedge (A \vee \neg B \vee \neg C).$$

Proving Equivalence of Formulas

A typical problem is the following: given two propositional formulas X and Y, are X and Y equivalent? That is, what is the truth value of the proposition $X \leftrightarrow Y$ (denoted by $X = Y$)? Roughly speaking, there are three ways to prove that X and Y are equivalent:

1. Use a truth table to check if the columns corresponding to X and Y are identical.
2. Apply a sequence of rules to X until Y appears. (We can also "work backwards" from Y and meet X at some formula "in the middle.")
3. Convert X and Y to DNF (or CNF) using the procedure described above, and check if the resulting formulas are identical. This is essentially a special case of Method 2.

Component Object Model & Object Linking and Embedding

COM is a framework for creating and using objects. COM, the Component Object Model delivers on the long-promised benefits of object technology: code reuse and off the shelf components. How does it do this? By providing a standard way to create and use components with a wide choice of tools, languages and applications.

The Component Object Model (COM) is a system technology that originated with Windows, but has begun to propagate to other platforms (the Macintosh, Compaq/Digital VMS, Compaq Digital Unix, Solaris, other Unix flavors, mainframes, etc.) as well. Its purposes for software developers are multi-fold. We'll focus on COM technology and concentrates on its advantages, which have made it one of the most versatile and indispensable technologies of today.

INTRODUCTION

OLE (Object Linking and Embedding)

OLE stands for object linking and embedding, is a feature that allows users to create and edit documents that contain objects created by different applications.

For example, you can embed bitmap images, sound clips, spreadsheet files, and other objects in Microsoft Word documents. The term "object linking and embedding" comes from these two types of actions for creating compound documents:

- **Linking** adds a link in a document that points to source data stored somewhere else. Linked objects are stored in the document as a path to the original linked data, usually a separate file from the container document. A linked object is a reference to information in another document. Link objects when you want to use the same information in more than one document. Then, if you change the original information, you need to update only the links in order to update the document containing the OLE objects. You can also set links to be updated automatically. When you link a drawing, you need to maintain access to the source application and the linked document. If you rename or move either of them, you may need to re-establish the link.

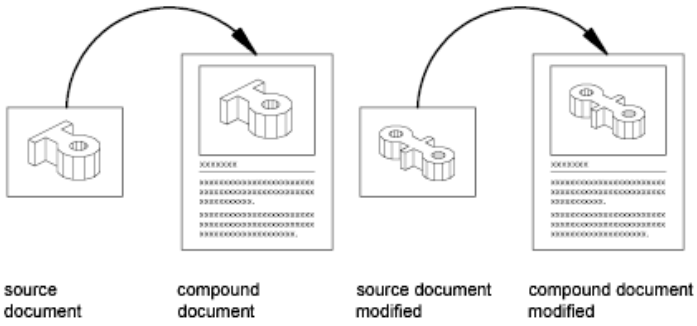


Figure 1. Linking

- **Embedding** on the other hand, adds one document directly to the other. Embedded objects are stored with the document that contains them. An embedded OLE object is a copy of information from another document. When you embed objects, there is no link to the source document and any changes made to the source document are not reflected in destination documents. Embed objects if you want to be able to use the application that created them for editing, but you do not want the OLE object to be

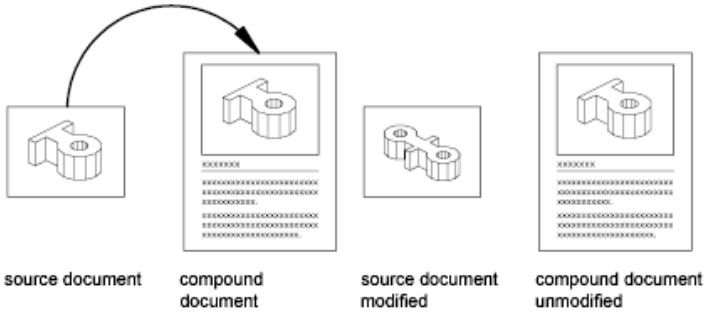


Figure 2 . Embedding

updated when you edit information in the source document.

Part of the confusion related to OLE technology comes from the fact that Microsoft has used this name for different purposes. Everything started with Object Linking and Embedding (OLE, for short) which was an extension of the DDE (Dynamic Data Exchange) model.

Using the Clipboard allows you to copy some raw data, and using DDE allows you to connect parts of two documents, Object Linking and Embedding allows you to copy the data from a server application to the client application, along with some information regarding the server, or a reference to some information stored in the Windows Registry. The raw data might be copied along with the link (Object Embedding) or kept in the original file (OLE Linking).

Its primary use is for managing compound documents, but it is also used for transferring data between different applications using drag and drop and clipboard operations. The concept of "embedding" is also used for embedding multimedia in Web pages, which tend to embed video, animation (include Flash animation), and music files within the HTML code.

OLE 1.0, released in 1991, was the first wildly adapted specification for developing component-document applications. It was the evolution of the original dynamic data exchange (DDE) concepts which Microsoft developed for earlier versions of Windows. While DDE was limited to transferring limited amounts of data between two running applications, OLE was capable of maintaining active links between two documents or even embedding one type of document within another.

OLE 1.0 later evolved to become architecture for software components known as the component object model (COM) which further graduated to Distributed Component Object Model (DCOM).

OLE 2.0 , Released in early 1993, it provided a much richer compound document model (i.e. the containing multiple data types like text, video, graphics etc.), as well as OLE automation, OLE drag and drop and generic services.

OLE 2, the second version of OLE, improved on OLE 1 and expanded the support for creating compound documents. OLE 2 was based on the model known as the Component Object Model (COM). Microsoft began to recognize that OLE 2 could be used to solve other software problems and that it could be applied to other areas of software development. Microsoft saw OLE 2 as an expandable architecture to create software and, as such, decided to drop the version number.

At the core of OLE 2.0 is the Component Object Model (COM), a specification that allows developers to design interfaces that enable interaction among components. In fact, OLE 2.0 is simply a set of COM interfaces designed by Microsoft.

ActiveX , In 1996, Microsoft renamed the OLE 2.0 technology as ActiveX. This version of OLE is commonly used by Web designers to embed multimedia files in Web pages. ActiveX is a set of programming technologies created by Microsoft that enables software components created in different languages to interact with one another in a networked environment. It evolved from the OLE development standard, which in recent years has expanded far beyond the concepts of object linking and embedding that formed the original acronym.

Like OLE, ActiveX is built on the COM programming model to support the full integration of software components. It supports Distributed COM (DCOM) for the transparent integration of those same components across distributed networks, including the Internet and intranets. However, ActiveX has been optimized for size and speed to allow developers to use subsets of the complex OLE interface to create highly interactive applications.

The terminology is changing as fast as the technology, and not all groups within Microsoft can agree on how to use the terms ActiveX and OLE. Think of ActiveX as something that was created when the "old" OLE collided with the Internet. ActiveX includes not only those Windows features built on COM family and the WinInet programming interface.

Overview

The Component Object Model (COM) is the foundation of much of the new Microsoft ActiveX technology, and after five years it's become an integral part of Microsoft Windows. Now we already have COM ++ and most Windows programming will involve COM, so you'd better start learning it now. But where do you begin? You could start with the Microsoft Foundation Class classes for ActiveX Controls, Automation, and OLE, but as useful as those classes are, they obscure the real COM architecture. You've got to start with fundamental theory, and that includes COM and something called an interface.
A component software architecture from Microsoft, which defines a structure for building program routines (objects) that can be called up and executed in a Windows environment.

Some parts of Windows and Microsoft developed applications are also built as COM objects. COM provides the interfaces between objects, and Distributed COM (DCOM) enables them to run remotely.

COM was designed with C++ programming environment in mind. It supports encapsulation, polymorphism, and reusability. However, COM was also designed to be operating at compatible at the binary level and therefore is different from a C++ object. Generally, the high level programming languages such as C, C++, PASCAL, and ADA are machine-dependent. As a binary object, a COM object concentrates on its interface with other objects. When not used in the environment of its creator, an interface is exposed that can be seen in the non-native foreign environment. It can be seen because it is a binary object and therefore *not machine-dependent*. This does not require the host environment or an interacting object to know anything about the COM object. It is important to note that COM is not a programming language; it is a binary standard that enables software components to interact with each other as objects. COM is not specific to any particular programming language. COM can work with any language that can support the binary layout of a COM object. It is a programming model to facilitate the programmability related to this standard.

What is COM?

The "problem" is that there's no standard way for Windows program modules to communicate with one another. "But," you say "what about the DLL with its exported functions, Dynamic Data Exchange (DDE), the Windows Clipboard, and the Windows API itself, not to mention legacy standards such as VBX and OLE 1? Aren't they good enough?" Well, no. You can't build an object-oriented operating system for the future out of these ad hoc, unrelated standards. With the Component Object Model, however, you can, and that's precisely what Microsoft is doing.

What's wrong with the old standards? A lot. The Windows API has too large a programming "surface area": more than 350 separate functions. VBXs don't work in the 32-bit world. DDE comes with a complicated system of applications, topics, and items. How you call a DLL is totally application-specific. COM provides a unified, expandable, objectoriented communications protocol for Windows that already supports the following features:

- A standard, language-independent way for a Win32 client EXE to load and call a Win32 DLL.
- A general-purpose way for one EXE to control another EXE on the same computer (the DDE replacement).
- A replacement for the VBX control, called an ActiveX control.
- A powerful new way for application programs to interact with the operating system.
- Expansion to accommodate new protocols such as Microsoft's OLE DB database interface.
- The distributed COM (DCOM) that allows one EXE to communicate with another EXE residing on a different computer, even if the computers use different microprocessor-chip families, that means different platform or hardware.

So what is COM? That's an easier question to ask than to answer. COM is a powerful integrating technology that allows you to mix all sorts of disparate software parts together at runtime. COM allows developers to write software that runs together regardless of issues such as thread-awareness and language choice. COM is a protocol that connects one software module with another and then drops out of the picture. After the connection is made, the two modules can communicate through a mechanism called an interface. Interfaces require no statically or dynamically linked entry points or hard-coded addresses other than the few general-

purpose COM functions that start the communication process. An interface (more precisely, a COM interface) is a term that you'll be seeing a lot of.

1. COM Objects

COM objects can be small or large. They can be written in several programming languages, and they can perform any kind of processing. A program can call the object whenever it needs its services. Objects can be run remotely (DCOM) over the network in a distributed objects environment.

2. Automation (OLE automation)

Standard applications, such as word processors and spreadsheets, can be written to expose their internal functions as COM objects, allowing them to be "***automated***" instead of being manually selected from a menu. For example, a script could be written to extract data from a database, summarize it and draw the graphics from a spreadsheet and place the results into a text document.

3. Controls (OLE controls, ActiveX controls)

Applications can invoke COM objects, called "controls," that blend in and become just a part of the program. ActiveX controls can also be downloaded from the Internet to make a Web page perform any kind of processing.

4. Compound Documents and ActiveX Documents

Microsoft's OLE compound documents are based on COM, which lets one document be embedded within or linked to another. ActiveX Documents are extensions to OLE that allow a Web browser, for example, to view not only Web pages, but any kind of document.

5. Programming Interfaces

Increasingly, Microsoft is making its standard programming interfaces conform to the COM object model so that there is continuity between all interfaces.

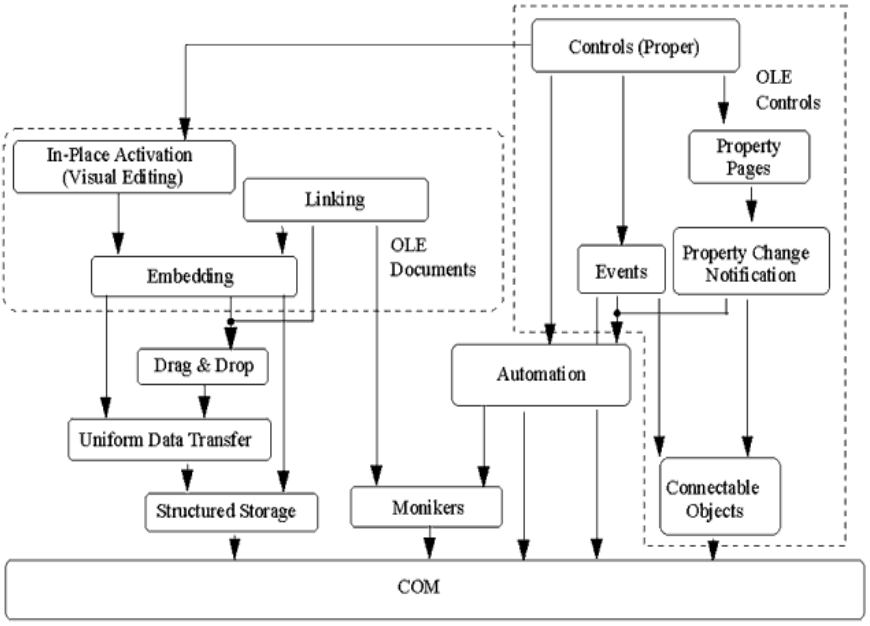


Figure 3. OLE technologies build on one another, with COM as the foundation.

THE ADVANTAGES AND DISADVANTAGES

ADVANTAGES

- The need to build and update entire application every time is eliminated. Rebuilding a single component is enough.
- It promotes component based software development which has several advantages like ability to use prepackaged components and tools from third party vendors into an application and support for code reusability in other parts of the same application.
- COM makes it possible for different language components, which adhere to COM specifications to interact with each other.
- It helps to access components loaded in different machines on the network.
- It can segregate applications via binary firewalls or interfaces that can reduce or eliminate dependencies between primary elements.
- It provides many features to developers in industry like language independency, binary standard, wide software industry support etc.

- **Object Oriented Features**

All the COM objects are Object Oriented as its name implies. There are three basic Object Oriented concepts:

i. **Encapsulation**: This is implemented by the interfaces provided by the COM Objects. The interface hides the implementation details from the end user and provides the functionality to the user.

ii. **Polymorphism**: This is often referred as "One Method Multiple Interfaces". A COM object can define a single method to perform a specific operation; but that operation could be implemented through various ways.

iii. **Inheritance**: When the user wants to incorporate some additional functionality to an existing COM object, he can enhance the existing COM object by inheriting a new COM object from it.

- **Loose Coupling** In software, which uses COM objects, one can easily replace an existing COM object with another COM object written in entirely another language as long as the signatures of the methods in both the COM objects remain same. In such a case, there will not be any change in the existing software code that uses the COM object.

- **Binary Language** Since most of the COM libraries are in binary language, it could be used by any application written in any language. So COM is language independent.

- **Resource utilization** Every COM object will be destroyed automatically as long as no client is using that object actively. This is implemented by COM using a technique called reference counting. Every COM object will maintain a reference count (the number of clients using that COM object); Once that count reaches zero (that means no clients are actively using that COM object), then that COM object will be destroyed automatically. With this approach, we can increase resource utilization in a single application. Resources such as memory will be best utilized by releasing the inactive/unused COM objects.

DISADVANTAGES

- **Difficult COM Component Development** COM components are often difficult for developers to implement. An extensive amount of code must be provided to implement a valid COM object. Furthermore, the software debugging tools for COM objects are less developed than their non-component counterparts. Since COM objects can be active on multiple computers, it is often difficult to determine the source of malfunction(s), if any.

- **Divergence in Standards** The COM architecture only specifies how software components communicate – not the specific interfaces they should use. Microsoft has already defined many interfaces that provide standard features such as drag-and-drop, data transfer, and persistent storage. These are not the only valid interfaces; any developer can define and implement their own custom built interface. Although this in one of the greatest strengths of COM, it is also one of its greatest weaknesses. With unchecked interface proliferation, developers will create their own closed set of interface standards that will be incompatible with the work of other developers. This problem has been partially averted through Microsoft's use of a standard scripting interface referred to as "IDispatch." The coordinated development of interface standards would be necessary for the ongoing success of COM.

- The other major drawbacks include difficulty in integrating internet technologies and expensive, difficult, and undependable deployment.

DCOM

Microsoft's Distributed COM (DCOM) extends the Component Object Model (COM) to support communication among objects on different computers—on a LAN, a WAN, or even the Internet. With DCOM, an application can be distributed at locations which are required by the customer and the application. Because DCOM is a seamless evolution of COM, one can take advantage of the existing investment in COM-based applications, components, tools, and knowledge to move into the world of standards-based distributed computing. In the process, DCOM handles low-level details of network protocols so one can focus on the real business thus providing great solutions to the customers.

COM+

It is the enhancement of the Microsoft Component Object Model (COM) that enables programmers to develop COM objects with more ease. For example, COM+ allows native C++ calls to be translated into equivalent COM calls. In addition, instead of defining COM interfaces in the traditional IDL language, they can be defined by more familiar programming syntax.

Where the Component Object Model (COM) ends, COM+ starts. COM+ extends COM to let you create components that scale better for:

- **Security in Distributed environments**: COM+ makes it easier to build components that can utilize the security subtleties of Distributed COM (DCOM) .
- **Component Services**: For event notifications, synchronization, and de-coupling of event sender and receiver (queued components, etc.).
- **Performance in Distributed environments**: COM+ makes it easier to build components that can stand the stress of high-volume instantiation requests over short periods of time.
- **Deployment in Distributed environments**: COM+ makes it easier to distribute server components or client-side components (typelibs and proxy-stub DLLs) without having to copy, unpack, register and configure each of them.

OLE DB Architecture

OLE DB stands for *Object Linking and Embedding Database*. Whereby linking it meant linking the object with the data source and by embedding it is meant to add the source data into the object. OLE DB is also an Application Programming Interface (API) that allows applications to access data from varied data sources. It is in fact a group of interfaces (APIs) that are used to gain access to data sources to fetch different kinds of data files like an object database, spreadsheets, personal databases, Indexed- sequential files, SQL-based DBMS files etc., which are not all necessarily SQL using file formats.

The OLE DB helps in facilitating data access for the application from the data sources. The access to the data source is possible through a set of abstractions or components in OLE DB namely; ***Data source, Session, Row sets and Command***. These four are also termed as OLE DB objects. By these four abstractions playing as a bridge between the application and data source, the client can gain access to varied (not only SQL) sources and types of data without getting into the technicalities of the process.

To fetch data from the data source, the OLEDB of an application first initializes the OLE. Followed by connection establishment with the data source. After this, a data access command is issued to be sent to the data source. Then the data sent as requested is processed and the OLE is uninitialized releasing the data source object. The OLE DB conceptually divided into two sections; Consumers and Providers. Consumers are the user applications that require the data and the providers are those who implement the APIs and in turn, fetch the required data from the data sources.

Data Source: SQL Server Native Client uses the term data source for the set of OLE DB interfaces used to establish a link to a data store, such as SQL Server. A data source object creates and manages the connection and contains permissions and authentications information.

Session Objects: A session manages a particular interaction with the data source to query and retrieve data. Each session is a single transaction. A transaction is an indivisible work unit defined by the ACID test (**Atomicity, Consistency, Isolation, Durability**). Session objects provide methods for creating commands and row sets and for creating and modifying tables and indexes. They also define transaction scope and can be used to create transaction objects, which are used to control nested transactions.

Rowsets and **Commands**: A rowset is a set of rows that contain columns of data. Rowsets are central objects that enable all OLE DB data providers to expose result set data in tabular form. Commands execute a text command such as a SQL statement. If the text command specifies a rowset, such as a SQL **SELECT** statement, the command creates the rowset.

Other Components Include:

- **Accessors** describes how data is stored in a consumer. It has a set of bindings (called a column map) between rowset fields (columns) and data members that you declare in the consumer.
- **Transactions** objects are used when committing or aborting nested transactions at other than the lowest level. A transaction is an indivisible work unit defined by the ACID test.
- **Enumerators** search for available data sources and other enumerators. Consumers that aren't customized for a particular data source use enumerator to search for a data source to use.
- **Errors,** any interface on any OLE DB object can generate errors.
- **Notifications** are used by groups of cooperating consumers sharing a rowset (where sharing means that the consumers are assumed to be working within the same transaction).

Distributed Database Management Systems And Data Allocation

The efficiency and performance of Distributed Database Management Systems (DDBMS) is mainly measured by its proper design and by network communication cost between sites. Fragmentation and distribution of data are the major design issues of the DDBMS. In this paper, we propose new approach that integrates both fragmentation and data allocation in one strategy based on high performance clustering technique and transaction processing cost functions. This new approach achieves efficiently and effectively the objectives of data fragmentation, data allocation and net- work sites clustering. The approach splits the data relations into pair-wise disjoint fragments and determine whether each fragment has to be allocated or not in the network sites, where allocation benefit outweighs the cost depending on high performance clustering technique. To show the performance of the proposed approach, we performed experimental studies on real database application at different networks connectivity. The obtained results proved to achieve minimum total data transaction costs between different sites, reduced the amount of redundant data to be accessed between these sites and improved the overall DDBMS performance.

Many researches have been done in the last few years in order to improve the performance of distributed data- base management systems (DDBMS). DDBMS is a collection of logically interrelated data that are physically allocated at different locations over a computer network. Most of recent researches are concerned about keeping the performance of DDBMS high so that they focused on how to design the database such that the over- all cost is kept minimal. Keeping the cost minimal is not an easy task as there are huge amount of transactions processing that increase the complexity of distributed databases. Several techniques have been proposed in order to improve database performance which can be achieved by improving at least one of the following database management issues: database fragmentation, data allocation and replication, and the network sites clustering.

The fragmentation process divides the database into portions each of which is called a fragment. Fragmentation can be horizontal, vertical or mixed. The main advantage of fragmentation is to improve the performance of distributed database design by increasing the efficiency since data is stored only where it is needed. Fragments can be allocated at different network sites in a process called data allocation.

The fragments allocation is an NP-complete problem so that the complexity is high. In order to reduce the complexity, some heuristic algorithms have been proposed to solve the problem. In the allocation process, each fragment is assigned to a network node and sometimes to more than one node to achieve the data availability, system reliability and performance.

The clustering technique is used for grouping distributed database network sites into logical clusters. In order to reduce the communication time for data allocation, there are many algorithms that are use to find the optimal solution for grouping distributed database network sites into a disjoint clusters and making a better data distribution among them. The clustering technique aims at eliminating the extra communication costs between the net- work sites and then enhancing the DDBMS performance.

Many existing algorithms of data fragmentation and allocation in DDBMS assume some restrictions on the number of network sites so that the results of such algorithms are impractical, and reflected on the efficiency and validity of their outcomes. Moreover, some constraints on network connectivity and transactions processing time will limit the applicability of the proposed solutions to a small number of DDBMS cases.

One of the drawbacks of fragmentation and allocation solutions is the high computational complexity of their associated algorithms. In fact, when distributing a database over a network with a big number of sites and then finding an efficient, reliable and optimal solution for fragmentation and allocation are considered difficult tasks.

This paper proposed a new technique that splits the database relations into disjoint fragments. In addition, it introduces a high speed clustering technique that groups the distributed network sites into a set of clusters according to their communication cost. Also, it proposes a new intelligent technique for data allocation and redistribution based on transactions processing cost functions. Moreover, it implements a user-friendly simulation tool that performs fragmentation, clustering, allocation and replication of a database, in addition to assisting database administrators in measuring DDBMS performance.

DDBMS (Distributed Database Management System)

A DDBMS (distributed database management system) is a centralized application that manages a distributed database as if it were all stored on the same computer. The DDBMS synchronizes all the data periodically, and in cases where multiple users must access the same data, ensures that updates and deletes performed on the data at one location will be automatically reflected in the data stored elsewhere.

Functions of Distributed Database System

Distribution basically leads to increased complexity in the system design and implementation. This is to achieve the potential advantages such as:

1. Network Transparencies
2. Increased Reliability
3. Improved Performance
4. Easier Expansion

Function of Centralized DBMS:

1. The basic function of centralized DBMS is that it provides complete view of our data.
   For example, we can have the query for the number of customers who are willing to buy worldwide.
2. The second basic function of Centralized DBMS is that it is easy to manage than other distributed systems.

The Distributed Database must be able to provide the following function in addition to those of a centralized DBMS's.

Functions of Distributed database system:

- **Keeping track of data**
  The basic function of DDBMS is to keep track of the data distribution, fragmentation and replication by expanding the DDBMS catalog.
- **Distributed Query Processing**
  The basic function of DDBMS is basically its ability to access remote sites and to transmits queries and data among the various sites via a communication network.
- **Replicated Data Management**
  The basic function of DDBMS is basically to decide which copy of a replicated data item to access and to maintain the consistency of copies of replicated data items.
- **Distributed Database Recovery**
  The ability to recover from the individual site crashes and from new types of failures such as failure of communication links.

- **Security**
  The basic function of DDBMS is to execute Distributed Transaction with proper management of the security of the data and the authorization/access privilege of users.
- **Distributed Directory Management**
  A directory basically contains information about data in the database. The directory may be global for the entire DDB, or local for each site. The placement and distribution of the directory may have design and policy issues.
- **Distributed Transaction Management**
  The basic function of DDBMS is its ability to devise execution strategies for queries and transaction that access data from more than one site and to synchronize the access to distributed data and basically to maintain the integrity of the complete database.

Database Partitioning

Various methods already exist describe data fragmentation in distributed DDBMS. Naturally, there are benefits and drawbacks to all schemes. Some methods need to incorporate performance evaluation ways, may not minimize the transactions response time and cannot guarantee the ability to process a given portion of a given transaction in all sites.

In our proposed approach, the database will be partitioned into pair-wise disjoint fragments by using a horizontal partitioning technique, in which the records of a relation split into disjoint fragments; this strategy guarantees the ability to processing all portions of a given transaction and distributes it precisely over the DDBMS sites.

Generating data fragments accomplished by performing the following processes respectively: defining transactions, creating segments and extracting disjoint fragments. Figure 1 below describes the architecture of the fragmentation method that supports the use of knowledge extraction and helps to achieve the effective use of small data packets.

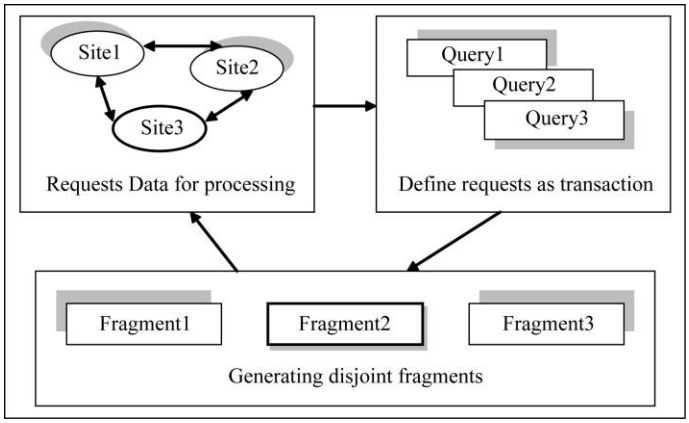


Figure 4 Generating disjoint fragments

As shown in Figure 1, the data request is initiated from the DDBMS sites (Site1, Site2, and Site3) and defines transactions as queries (Query1, Query2, and Query3) and then these transactions (queries) are processed into disjoint fragments (Fragment1, Fragment2, and Fragment3). The database transaction could be associated with more than one relation, in this case, the transaction should be divided into a number of sub-transactions equal to the number of relations used that transaction. The process of generating fragments is described as follows:

1. Database fragmentation starts with any two fragments having intersection records between them. If there is an intersection, then three disjoint fragments will be generated as follows:
   - The common records in the two intersected fragments,
   - The records in the first fragment but not in the second segment, and
   - The records in the second fragment but not in the first segment.
2. The intersecting fragments are then removed from the fragments list. This process continues until removing all the intersecting fragments.
3. The new derived fragments and the non-overlapped ones that do not intersect with any other fragments from the new list of totally disjoint fragments.

Network Sites Clustering

The benefit of generating database disjoint fragments can't be completed unless it enhances the performance of the distributed database system. As the number of database sites becomes too large, a problem of supporting high system performance with consistency and availability constraints becomes crucial. Different techniques could be developed for this purpose; one of them consists of clustering distributed networking sites. Clustering database sites is a technique in which the sites that have similar physical property (e.g., having comparable communication costs) are logically grouped together in order to increase the performance of the distributed database system. However, grouping sites into clusters is still an open problem and it is proven that the optimal solution to this problem is NP-Complete since it is transformed to a cheapest path problem. Therefore, near-optimal solution for grouping database sites into clusters helps to eliminate the extra communication costs between the sites during the process of data allocation and improves the system performance. Performing sites grouping after database fragmentation, will speed up the process of data allocation by distributing the fragments over clusters of sites rather than site by site. Thus, the communication costs are minimized and the distributed database system performance is improved.

Clustering Algorithms

Since the task of clustering is subjective, the means that can be used for achieving this goal are plenty. Every methodology follows a different set of rules for defining the '*similarity'* among data points. In fact, there are more than 100 clustering algorithms known. But few of the algorithms are used popularly, let's look at them in detail:

- **Connectivity models:** As the name suggests, these models are based on the notion that the data points closer in data space exhibit more similarity to each other than the data points lying farther away. These models can follow two approaches. In the first approach, they start with classifying all data points into separate clusters & then aggregating them as the distance decreases. In the second approach, all data points are classified as a single cluster and then partitioned as the distance increases. Also, the choice of distance function is subjective. These models are very easy to interpret but lacks scalability for handling big datasets. Examples of these models are hierarchical clustering algorithm and its variants.

- **Centroid models:** These are iterative clustering algorithms in which the notion of similarity is derived by the closeness of a data point to the centroid of the clusters. K-Means clustering algorithm is a popular algorithm that falls into this category. In these models, the no. of clusters required at the end have to be mentioned beforehand, which makes it important to have prior knowledge of the dataset. These models run iteratively to find the local optima.

- **Distribution models:** These clustering models are based on the notion of how probable is it that all data points in the cluster belong to the same distribution (For example: Normal, Gaussian). These models often suffer from overfitting. A popular example of these models is Expectation-maximization algorithm which uses multivariate normal distributions.

- **Density Models:** These models search the data space for areas of varied density of data points in the data space. It isolates various different density regions and assign the data points within these regions in the same cluster. Popular examples of density models are DBSCAN and OPTICS.

Fragments Allocation

Data allocation and replication technique places and distributes the database fragments on the clusters and their respective sites. Initially fragments are allocated to the clusters that have transactions using that fragments. Allocating fragments to a small number of clusters instead of large number of sites will reduce the number of communications and therefore enhance the system performance. Figure 2 illustrates the structure of data allocation and replication technique.

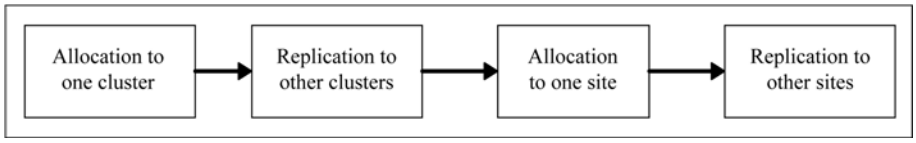


Figure 5 Data Allocation and Replication Technique

A heuristic fragment allocation and replication technique will be introduced to perform the processes of frag- ments allocation in the distributed database system. Initially, all fragments are subject for allocating to all clus- ters having transactions using these fragments at their sites. If the fragment shows positive allocation decision value (i.e. allocation benefit greater than or equal to zero) for a specific cluster, then the fragment is subject for allocating at each site in this cluster, otherwise the fragment is not allocated (cancelled) from this cluster. This step is repeated for each cluster in the distributed database system.

As a result of the previous step, the fragment that shows positive allocation decision value at any cluster is a candidate for allocating at all sites of this cluster. If the fragment shows positive allocation decision value at a site of cluster that already shows positive allocation decision value, then the fragment is allocated at this site, otherwise, the fragment is not allocated to this site. This step is repeated for each site at this cluster.

To ensure data availability in the distributed database system, each fragment should be allocated to at least one cluster and one site. In case a fragment shows negative allocation decision value at all clusters, the fragment is allocated to the cluster that holds the least average communication cost and then to the site that has the least communication cost in this cluster.

Allocation Cost Functions: The allocation cost functions identifies the allocation status which is computed as a logical value for the com- parison between the cost of remote access the fragment to the cluster/site and the cost of allocating the fragment to the cluster/site. If the cost of remote access the fragment to the cluster/site is greater than or equals to the cost of allocating the fragment to the cluster/site, then the allocation status is positive and the fragment is allocated to the cluster/site. On the other hand, if the cost of remote access is less than the cost of allocating, then the alloca- tion status is negative and the fragment is cancelled from the cluster/site.

Performance Evaluation

We believe that the performance of allocation and network load are the major performance issues; hence, we will focused our performance evaluation on these two issues as they will maximize the overall system through- put at each network site in the distributed database environment, and minimize the cost of data that has been transferred and processed.

Fragment Allocation Performance Evaluation

The technique of fragment allocation and replication at clusters is evaluated according to the performance generated by reducing the size of fragments that allocated finally at the clusters. The closest methods in the literature to the proposed technique of fragment allocation and replication are those proposed above. The main differences between these two methods are described as follows.

Compared to the allocation techniques in the literature, our allocation and replication technique considers different communication costs, namely, the cost of the real network communication between cluster sites, the up- date and retrieval costs (*i.e*., it mostly represents the cost of writing operation that takes place during the execution time). Moreover, with our clustering technique, database sites are grouped according to a clustering range and not only to a specific communication cost. Our clusters can communicate with each other instead of prefer- ring to have all fragments in their sites. This communication is cheaper than allocating fragments in all sites. In addition to the gain in term of communication cost, the independency of our clusters makes our DDBMS more reliable and more functional.

Network Performance Evaluation

The DDBMS network workload involves the queries from a number of database users who access the same data- base simultaneously. The amount of data needed per second and the time in which the queries should be processed depend on performance of the database applications running under DDBMS network.

- **Server Load** The server load determines how much the speed of the server in terms of bits per seconds. The following figures depict the network server loads in the proposed DDBMS.
- **Network Delay** The network delay is the delay caused by the transactions traffic on the server of the distributed database system. The maximum time required for the network system to reach the steady state is defined as network delay, it’s measured in millisecond.

The Four Types of JDBC drivers

A Driver is a program that controls a device for it to accomplish a task. *Java database Connectivity* (JDBC) allows for accessing any form of tabular data from any source. Relational databases are most common, but JDBC drivers for XML, Excel, or legacy data sources can be obtained.

JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates. We have divided the implementation types into four categories:

1. JDBC-ODBC Bridge Driver:

The JDBC-ODBC bridge driver uses ODBC (*Open Database Connectivity*), a standard or open application programming interface (API) for accessing a database, driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. They are drivers that implement the JDBC API as a mapping to another data access API, such as Open Database Connectivity (ODBC). Drivers of this type are generally dependent on a native library, which limits their portability. It can be easily connected to any database.

The Performance is degraded because JDBC method call is converted into the ODBC function calls. And also, the ODBC driver needs to be installed on the client machine.

2. Native-API driver:

Drivers that are written partly in the Java programming language and partly in native code. The drivers use a native client library specific to the data source to which they connect. Because of the native code, their portability is limited. The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native C/C++ API calls, which are unique to the database. It is not written entirely in java. Their performance upgraded compared to the JDBC-ODBC bridge driver.

The Native driver and the Vendor client library need to be installed on each client machine.

3. Network-Protocol driver (middleware driver):

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java. This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases. Your application server might use a Type 1, 2, or 4 drivers to communicate with the database, understanding the nuances will prove helpful.

No client-side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc. Network support is required on client machine and requires database-specific coding to be done in the middle tier, thus maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

4. Database-Protocol driver (Pure Java driver):

In a Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself. This kind of driver is extremely flexible, you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.

Resource Description Framework, Extensible Markup Language and Ontology

Resource Description Framework (RDF)

The Resource Description Framework (RDF) is a framework for representing information in the Web. RDF is a general framework for describing website metadata, or "information about the information" on the website. It provides interoperability between applications that exchange machine-understandable information on the Web. RDF details information such as a site's sitemap, the dates of when updates were made, keywords that search engines look for and the Web page's intellectual property rights.

The Resource Description Framework (RDF) is an infrastructure that enables the encoding, exchange and reuse of structured metadata. RDF is an application of XML that imposes needed structural constraints to provide unambiguous methods of expressing semantics. RDF additionally provides a means for publishing both human-readable and machine-processable vocabularies designed to encourage the reuse and extension of metadata semantics among disparate information communities. The structural constraints RDF imposes to support the consistent encoding and exchange of standardized metadata provides for the interchangeability of separate packages of metadata defined by different resource description communities.

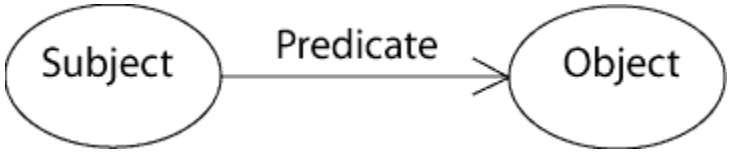
RDF defines a simple, yet powerful model for describing resources. A syntax representing this model is required to store instances of this model into machine-readable files and to communicate these instances among applications. XML is this syntax. RDF imposes formal structure on XML to support the consistent representation of semantics.

The underlying structure of any expression in RDF is a collection of triples, each consisting of a subject, a predicate and an object. A set of such triples is called an RDF graph. Each triple represents a statement of a relationship between the things denoted by the nodes that it links. Each triple has three parts:

A. Subject,

B. Object, and

C. Predicate (also called a property) that denotes a relationship.



The direction of the arc is significant: it always points toward the object. The nodes of an RDF graph are its subjects and objects. The assertion of an RDF triple says that some relationship, indicated by the predicate, holds between the things denoted by subject and object of the triple.

Extensible Markup Language (XML)

XML is a markup language for documents containing structured information. A markup language is a mechanism to identify structures in a document. The XML specification defines a standard way to add markup to documents.

Structured information contains both content (words, pictures, etc.) and some indication of what role that content plays (for example, content in a section heading has a different meaning from content in a footnote, which means something different than content in a figure caption or content in a database table, etc.). Almost all documents have some structure.

XML specifies neither semantics nor a tag set like HTML. In fact XML is really a meta-language for describing markup languages. In other words, XML provides a facility to define tags and the structural relationships between them. Since there's no predefined tag set, there can't be any preconceived semantics. All of the semantics of an XML document will either be defined by the applications that process them or by stylesheets.

Extensible, XML is extensible. It lets you define your own tags, the order in which they occur, and how they should be processed or displayed. Another way to think about extensibility is to consider that XML allows all of us to extend our notion of what a document is: it can be a file that lives on a file server, or it can be a transient piece of data that flows between two computer systems (as in the case of Web Services).

Markup, The most recognizable feature of XML is its tags, or elements (to be more accurate). In fact, the elements you'll create in XML will be very similar to the elements you've already been creating in your HTML documents. However, XML allows you to define your own set of tags.

Language, XML is a language that's very similar to HTML. It's much more flexible than HTML because it allows you to create your own custom tags. However, it's important to realize that XML is not just a language. XML is a meta-language: a language that allows us to create or define other languages. For example, with XML we can create other languages, such as RSS, MathML (a mathematical markup language), and even tools like XSLT.

Ontology

An ontology is a formal description of knowledge as a set of concepts within a domain and the relationships that hold between them. To enable such a description, we need to formally specify components such as individuals (instances of objects), classes, attributes and relations as well as restrictions, rules and axioms. As a result, ontologies do not only introduce a sharable and reusable knowledge representation but can also add new knowledge about the domain.

There are, of course, other methods that use formal specifications for knowledge representation such as vocabularies, taxonomies, thesauri, topic maps and logical models. However, unlike taxonomies or relational database schemas, for example, ontologies express relationships and enable users to link multiple concepts to other concepts in a variety of ways.

In computer science and information science, an ontology encompasses a representation, formal naming and definition of the categories, properties and relations between the concepts, data and entities that substantiate one, many or all domains of discourse.

Every field creates ontologies to limit complexity and organize data into information and knowledge. As new ontologies are made, their use hopefully improves problem solving within that domain. Translating research papers within every field is a problem made easier when experts from different countries maintain a controlled vocabulary of jargon between each of their languages.

Ontology Web Language (OWL)

The Ontology Web Language (OWL) was developed—or, more accurately, was derived from several earlier language initiatives—in order to provide a standardized way of representing ontologies on the semantic web.

OWL is a semantic web computational logic-based language, designed to represent rich and complex knowledge about things and the relations between them. It also provides detailed, consistent and meaningful distinctions between classes, properties and relationships. By specifying both object classes and relationship properties as well as their hierarchical order, OWL enriches ontology modelling in semantic graph databases, also known as RDF triples. Also, OWL comes equipped with means for defining equivalence and difference between instances, classes

and properties. These relationships help users match concepts even if various data sources describe these concepts somewhat differently. They also ensure the disambiguation between different instances that share the same names or descriptions.

RDF vs XML

RDF and XML both attempt to address the problem of enabling different programs and computers to communicate effectively with each other. XML is a syntax and its data model is a tree. RDF is a data model based on a graph, that uses URIs and has several different syntax, including an XML syntax. Nevertheless, both XML and RDF can be used to represent structured data on the web and move data around between applications.

There are different ways of representing data in XML. In a way, the tree data model of XML is too flexible: you can say what you want to say in so many different ways. In order to understand what the XML document is about, an application needs the DTD (Document Type Definition) or the schema. RDF also has a schema, namely an ontology, represented in RDFS (RDF Schema) or OWL (Web Ontology Language). The ontology adds the semantics to your data and even allows you to infer new information from your current data (XML can't do this).

The complexity of querying the XML tree is because there are in general a large number of ways in which the XML maps onto the logical tree, and the query you write has to be independent of the choice of them. So much of the query is an attempt to basically convert the set of all possible representations of a fact into one statement. This is just what RDF does. It gives you some standard ways of writing statements so that however it occurs in a document, they produce the same effect in RDF terms. The same RDF tree results from many XML trees.

RDF vs RDFS vs OWL

RDF: Straightforward representation, focused on the instances and on the mapping to their types (`rdf:type`). It is possible to define custom properties to link data and creating triples. RDF data are queried with SPARQL.

RDFS: Some situations are not easily modelled by RDF alone, it is sometimes interesting to represent more complex relations like subclasses (*the type of a type*) for example. RDFS provides special means to represent such cases, with constructs like `rdfs:subClassOf`, `rdfs:range` or `rdfs:domain`.For instance if you have the triples John a Man and Man `rdfs:subClassOf` Human then you should generate as well the triple John a Human. Note that this is not possible to do with RDF alone.

OWL: The highest level of expressivity. Relation between classes can be formally modelled based on description logics (mathematical theory). it is possible to express complex constructs such as chained properties for instance or restriction between classes. OWL serves to build ontologies or schema on the top of RDF datasets. OWL adds ontological capability to RDF (which on its own only provides extremely limited capability for formal knowledge representation), by providing the apparatus to define the components of your triple using formal computable first order description logic. In effect, this means that you can use an OWL ontology as a schema for RDF

ONLINE RESOURCES & REFERENCES

- http://www.microsoft.com/technet
- http://computing-dictionary.thefreedictionary.com (Definitions and Figures)
- http://www.howtodothings.com/computers/a1239attributesadvantages-of-com.html
- http://www.peterindia.net/COMOverview.html
- http://www.richardhaleshawgroup.com
- http://media.wiley.com/product_data/excerpt/23/07645599/0764559923.pdf
- http://www.emory.edu/BUSINESS/et/com/
- http://en.wikipedia.org/wiki/Object_linking_and_embeddi ng (OLE reference
- https://www.cambridgesemantics.com/blog/semantic-university/learn-owl-rdfs/rdfs-vs-owl/
- https://www.quora.com/What-is-the-difference-between-RDF-and-OWL
- https://www.iro.umontreal.ca/~lapalme/ForestInsteadOfTheTrees/HTML/ch07s04.html
- https://www.cambridge.org/core/books/semantic-web-explained/web-ontology-language/D1A64A6D0591D33B5FCB080D9F6CD55A
- https://dl.acm.org/doi/10.1145/1295289.1295290
- https://www.sciencedirect.com/topics/computer-science/web-ontology-language
- https://blog.ldodds.com/2012/06/12/principled-use-of-rdfxml/
- http://www.dlib.org/dlib/may98/miller/05miller.html
- https://www.researchgate.net/post/What_is_the_difference_between_RDF_RDFS_and_OWL
- https://en.wikipedia.org/wiki/Ontology_(information_science)
- https://www.dataversity.net/introduction-to-rdf-vs-xml/