International Conference on Control, Engineering & Information Technology (CEIT'14)
Proceedings - Copyright IPCO-2014, pp. 278-296
ISSN 2356-5608

Intelligent Implementation Processor Design for Oracle Distributed Databases System

Hassen Fadoua, Grissa Touzi Amel

[1]Université Tunis El Manar, LIPAH, FST, Tunisia

[2]Université Tunis El Manar, ENIT, LIPAH,FST, Tunisia

{hassen.fadoua@gmail.com;amel.touzi@enit.rnu.tn}

Abstract. Despite the increasing need for modeling and implementing Distributed Databases (DDB), distributed database management systems are still quite far from helping the designer to directly implement its BDD. Indeed, the fundamental principle of implementation of a DDB is to make the database appear as a centralized database, providing series of transparencies, something that is not provided directly by the current DDBMS. We focus in this work on Oracle DBMS which, despite its market dominance, offers only a few logical mechanisms to implement distribution. To remedy this problem, we propose a new architecture of DDBMS Oracle. The idea is based on extending it by an intelligent layer that provides: 1) creation of different types of fragmentation through a GUI for defining different sites geographically dispersed 2) allocation and replication of DB. The system must automatically generate SQL scripts for each site of the original configuration.

Keywords: distributed databases; big data; fragmentation; allocation; replication.

1 Introduction

The organizational evolution of companies and institutions that rely on computer systems to manage their data has always been hampered by centralized structures already installed, this architecture does not respond to the need for autonomy and evolution of the organization because it requires a permanent return to the central authority, which leads to a huge waste of time and an overwhelmed work. Moreover, the world digitalization generated a new information phenomenon named "Big Data". Huge amounts of information created and exchanged in various formats and structures made impossible the classic processing methods in centralized architecture. Even with the outstanding progress of computers and networks, no known centralized architecture pretends to handle this scale. The example of Google 2012 migration is one the most expressive examples. The market leader of Big Data processing, invested in distributed database architectures and migrated some of its sensitive data to F1, the fault tolerant distributed database. In the few released paper, Google describes this internally developed RDBMS, F1, as a natively distributed database. Thus, the requirement of a database management system that can handle these "data

islands" in harmony with centralized system features is the first of the motivations that underlie the development of distributed databases. The second motivation behind RDBMS evolution is splitting data between network interconnected nodes to handle Big Data concept and afford convenient resources to process huge and complex incoming information amounts. Regarding very large volumes of data, splitting improves dramatically queries performance and speeds up database maintenance as huge volumes and models of data can be delegated to multiple nodes and processed in a parallel policy.

Unfortunately, existing DDBMS have several limitations. The world's leading providers of DBMS such as Oracle, offers only few partitioning commands in console mode. The tool for the implementation of Oracle DDB, known as Oracle Partitioning [9], remains limited towards end database administrators' expectations. This tool has several limitations: 1) it does not offer a GUI to ease DBA tasks in data distribution related tasks, and 2) there is no direct methods to assist top-down migration approach from the centralized database to the distribution strategy. The designer must describe the DDB necessary scripts to manually distribute its data to produce results consistent with its distributed design 3) It cannot handle automatically cross sites referential integrity constraints 4) It does not cover the calculations and updates requiring access to multiple sites at once.

Consequently, the design and implementation of a DDB has never been an easy task especially when dealing with huge database schemes. In this context, Rim's [10] and Hassen's[5] work can be mentioned. Their work proposed an expert system to help designing the DDB. These tools are more concerned to suggest distribution of data on different sites, regardless of the heavy task left to the designer to implement this DDB on different sites or the validation process of fragmentation if the user decides to change its design in response to new needs.

In this paper, we propose a new approach to assist design and implementation of DDB. This approach was validated through the design and implementation of an assistance tool that provides a graphical interface for different types of fragmentation, allocation and duplication along with validation at each step of the process. Then, the system automatically generates SQL scripts for each site of the initial configuration. We prove that the proposed tool can be integrated as an intelligent layer to any existing DDBMS.

Besides this introduction, this paper includes five sections. Section 2 presents the basic concepts of DDBS. Section 3 presents an example of implementation of DDB, illustrating the problems and limitations of existing DDBMS. Section 4 presents the motivation behind this work. Section 5 presents the new approach for Oracle DDBS. Section 5 presents the approach validation by describing the created tool. Section 6 makes an evaluation of this work and gives some future perspectives of it.

2 Base concepts

2.1 Distributed Database (DDB)

A distributed database is a collection of logically interconnected data physically stretched over a network.

2.2 Distributed database management system (DDBMS)

A distributed database management system is a software that manages a distributed database and ensures its transparency towards users.

Most important DDBMS on the market are Oracle, MySql, INGRES[12](currently ACTIAN), CASSANDRA and Informix.

In 1987, DATE [3] established twelve rules to define the perfect DDBMS, based on a fundamental principle: User must see DDBMS as a non distributed database management system. The twelve rules are: Local autonomy, no dependency to a central site, continuous availability, Location independence, Fragmentation independency, Replication independence, Distributed query processing, Distributed transaction processing, Hardware abstraction, Operating systems independence, Network independence and database independence.

2.3 Transparency levels

DDB implementation must comply with a series of transparencies. This set of transparencies' main aim is to let end-user see a DDB as a centralized database. Those levels can be described as:

- Distribution transparency ensures that end-user can ignore data replication or splitting. As a direct consequence, on data update, the system takes care of updating all the copies of the altered row.
- Transaction transparency grants overall database transparency on concurrent users' access to data and on system faults.
- Performance transparency which provides a reliable query management even when referencing data on multiple sites.
- Query transparency on mixed content call from different sites in the same call.
- DBMS transparency which ensures access to different DBMS without user being aware of it.

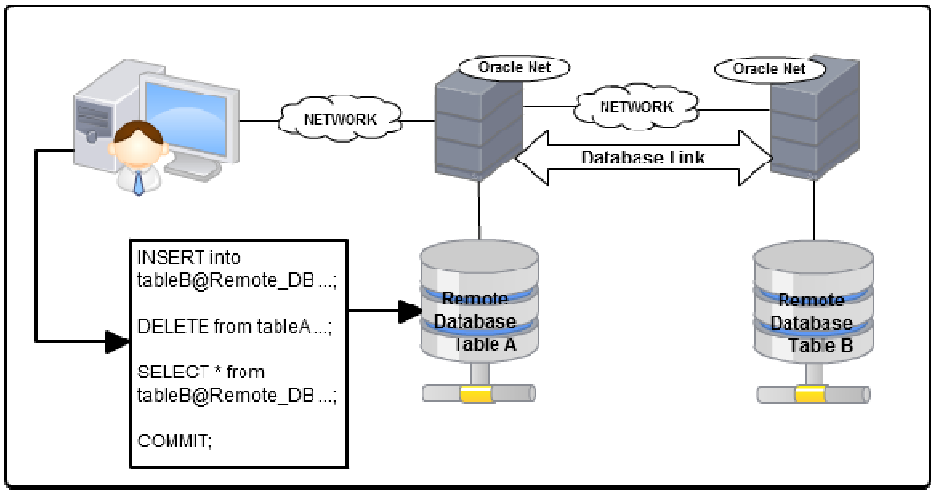
3 Summary of Oracle Distributed Database System

Oracle announced distributed DBMS capabilities in 1987, but largely as marketing ploy. The first Oracle product to reasonably support distributed database processing is Oracle 7, which has been in the market since 1993. The official support for Oracle

distributed concepts began with Oracle 8.0 in 1997. Oracle latest version, 12c, is said to be enhanced with its 10th generation Oracle Partitioning tool.
In its latest version, Oracle 12c offers three basic partitioning strategies: By range, by discrete list (for a region column as a partitioning key, 'North America' may contain values 'Canada', 'USA' or 'Mexico'), or by an internal hash algorithm.
A distributed database appears to a user as a single database but is, in fact, a set of databases stored on multiple computers. The data on several computers can be simultaneously accessed and modified over a network access. Each database server in the distributed database is controlled by its local DBMS, and each one cooperates to maintain the consistency of the global database.
Oracle supports two types of distributed databases. In a system based on homogeneous distributed data, all databases are Oracle databases. In a system of distributed heterogeneous database, at least one of the databases is not an Oracle database.

3.1 Client/Server Architecture:

Oracle database server is the software that manages Oracle databases, and client applications that interact with the server to add, update, retrieve or delete data.
Each computer in the network can host one or more databases. Databases are connected together through database links which are unidirectional. Figure 2 shows an example of connection to a remote database table from a local database. This is done transparently through Oracle DB links.



links.

Figure 1 shows the two types of queries that are sent to database to retrieve data: A direct request that caters to the local DB and the second query type, indirect, that manipulates remote database. All these requests are part of a transaction that can

either be committed or rolled back. This mechanism is called the two-phase validation.

3.2 Allocation mechanism in Oracle

Like any commercial DDBMS, Oracle does not accept the fragmentation mechanism type, although the administrator can manually allocate data data to produce similar results. This has the effect of shifting the responsibility under the auspices of the end user, who must know that a table has been fragmented and convert this knowledge into the application. In other words, the Oracle DDBMS does not ensure transparency of the distribution, while it allows location transparency [8].

Oracle RDBMS offers the following tools to create a DDB:

1. **Databases inter-connection**: Oracle DDB are built on database connections or links, named, in Oracle terminology, DBLINK. A database link is a pointer that defines a one-way communication path from an Oracle Database server to another database server. The link pointer is actually defined as an entry in a data dictionary table. To access the link, you must be connected to the local database that contains the data dictionary entry. A database link connection is one-way in the sense that a client connected to local database A can use a link stored in database A to access information in remote database B, but users connected to database B cannot use the same link to access data in database A. If local users on database B want to access data on database A, then they must define a link that is stored in the data dictionary of database B. A database link connection allows local users to access data on a remote database. For this connection to occur, each database in the distributed system must have a unique global database name in the network domain. The global database name uniquely identifies a database server in a distributed system.

2. **Location transparency**: After dblinks creation, Oracle objects such as views, synonyms and stored procedure can be used to hide data distributed aspect towards end-users and applications. A synonym in Oracle lexicon is an alias that may point to a remote schema, and even reference an object inside the remote schema. A table T1 in a site S1 where schema SC1 is defined as a dblink DS1 may be referenced as T1DS1 from another site, where T1DS1 is actually “SC1”.”T1”. Thereby, in user scripts the remote table T1 is called in the same way as a local table. Views can be defined likewise to reference remote object in the same compiled view.

3. **Data replication**: In order to reduce exchanged amounts of data among the network, and consequently improve queries’ performance, Oracle offers multiple options in basic replication features such as Sql Plus COPY command and materialized views. COPY command is reported to be obsolete in further releases and may not support new data types. This command copies data from a query to a table in a local or remote database in create, append, insert and replace mode. A materialized view contains a complete or partial copy of a target master from a

single point in time. The target master can be either a master table at a master site or a master materialized view at a materialized view site. A master materialized view is a materialized view that functions as a master for another materialized view. A multitier materialized view is one that is based on another materialized view, instead of on a master table. In the advanced replication features, Oracle implements multi-master (peer-to-peer) and asynchronous methods.

In order to ensure transparency, designer must stick to following steps:

1. User accounts creation among sites.
2. Bi-directional links creation between different sites (using oracle CREATE DBLINK command).
3. Local schema implementation on each site
4. Synonyms definition to ensure location transparency
5. View and/or materialized views creation and/or snapshots to insure fragmentation independent schema. On each materialized view and snapshot definition, we have to specify update mode (asynchronous, synchronous) and refresh delay in accordance with application need.
6. Stored procedure definition, as PL/SQL script, on each update operation in a way to make data fragmentation and duplication automated and transparent.
7. As a DBMS can ensure only local data integrity, designer must define PL/SQL triggers that allow checking distributed data integrity among DDB.

4 Sample DDB implementation under Oracle

In this section, a part of the distributed database implementation is described via a real Oracle 10g script to highlight the designer task complexity.

Example

Three institutions of the University of Tunis El Manar: National Engineering School of Tunis (ENIT), Faculty of Mathematical, Physical and Natural Sciences of Tunis (FST) and Faculty of Economic Sciences and Management of Tunis (FESMT) have decided to pool their libraries and service loans, to enable all students to borrow books in all the libraries of the participating institutions. Joint management of libraries and borrowing is done by a database distributed over 3 sites (Site1 = ENIT, Site2 = FST and Site3 = FESMT), the global schema is as follows:

Table 1. Sample centralized database schema

EMPLOYEE (**SSN**,emp_ fname, emp_ lname, Address, Status, Assignment)
STUDENT (**NCE**, stud_fname, stu_lname, Address, Institution, Class, nb_borrow)
BOOK (**Id_book**, Title, editor, Year, Area, Stock, website)
AUTHOR (**Id_book**, au_lname, au_fname)
LOANS (**Id_book**, NCE, **date_borrows**, return_date)

The management of this application is based on the following assumptions:

1. An employee is assigned to a single site. Each site is responsible of managing its employee.
2. A student is enrolled in a single institution, but can borrow from all libraries.
3. A book borrowed from a library is rendered to the same library.
4. The nb_borrow field of STUDENT relation is used to limit the number of books borrowed by a student simultaneously on all libraries. It is updated at each loan and each return, regardless of the lending library.
5. Each institution manages its own students.
6. Each library manages its staff and works it holds.

The first conclusion we can deduct is that overall relations must be split and distributed over different sites. The second inference of functional hypothesis is to split STUDENT table into two vertical fragments and duplicate the vertical fragment created on every site. The table fragment to duplicate is the sub-relation containing student id (NCE) and the numbers of active borrows (nb_borrow).
The resulting local schema for site FST for example would be:

Table 2. Local schema description

D language script: Local Schema for site FST
EMPLOYEE_FST (**SSN**,emp_ fname, emp_ lname, Address, Status, Assignment) **STUDENT_FST (NCE, stud_fname, stu_lname, Address, Institution, Class)** **STUDENT_BIBLIO (NCE, nb_borrow)** BOOKS_FST (**Id_book,** Title, editor, Year, Area, Stock, website) AUTHORS_FST (**Id_book**, au_lname, au_fname) LOANS_FST (**Id_book**, NCE, **date_borrows**, return_date)

The main advantages of the described distribution policy are:

- Reducing exchanged data size among sites when requesting students' data between sites.
- Lowering network activity by local checks on active borrows (nb_borrows) as it will be available locally thanks to STUDENT_BIBLIO fragment replication.

This distribution policy isn't the perfect strategy because of the more expensive updates on STUDENT_BIBLIO.

To illustrate the necessary very long task for implement our DDB, we describe in the following listing a part of the necessary script for implementing the relation STUDENT.

Table 3. Local schema creation script

PL/SQL Script: An excerpt from site creation script
---------- -- DDL for DB Link BB1 ---------- CREATE DATABASE LINK "ENIT_dblink" CONNECT TO "ROOT" IDENTIFIED BY VALUES 'root'

```
  USING '(DESCRIPTION= (ADDRESS=   (PROTOCOL=TCP)   (HOST=127.0.0.1)
(PORT=1521))  (CONNECT_DATA=   (SERVICE_NAME=ENIT)))';
--------------------------------------------------------
--  DDL for Table STUDENT_FST
--------------------------------------------------------
  CREATE TABLE  STUDENT_FST  ( NCE  NUMBER, ST_FNAME
VARCHAR2(200),
ST_FLNAME  VARCHAR2(200), ADRESS VARCHAR2(200 CHAR),
CLASS  NUMBER, CURSUS  VARCHAR2(200 CHAR),Constraint PK11 primary key
(NCE)  ) ;
--------------------------------------------------------------------------------
--  DDL for Synonym ENIT
--------------------------------------------------------------------------------
CREATE SYNONYM STUDENT_ENIT FOR STUDENT_ENIT@ENIT;
--------------------------------------------------------------------------------
--  DDL for materialized view STUDENT
--------------------------------------------------------------------------------
create materialized view STUDENT
refresh complete start with sysdate  next sysdate + 7
 as
(SELECT * from FOR STUDENT_ENIT@ENIT) UNION
(SELECT * from FOR STUDENT_ENIT@FST) UNION
(SELECT * from FOR STUDENT_ENIT@FESMT)
--------------------------------------------------------------------------------
--  DDL for trigger insetSudent
--------------------------------------------------------------------------------
CREATE OR REPLACE trigger insertStudent
before insert on Student
for each rowdeclare
excep exception ;
nbTuples number ;
begin
nbTuples :=0;
select count (*) into nbTuples from Etudiant where
NCE=:new. NCE ;
if ( nbTuples != 1) then raise excep ;
else
IF : new .Institution := "ENIT" THEN
INSERT INTO Etudiant_ENIT (NCE,ST_FNAME, ST_LNAME, ADRESS,
Institution,"CLASS", CURSUS) VALUE
( : new . NCE , : new. ST_FNAME,: new. ST_LNAME, : new . adress ,
 new . Institution,: new . "CLASS" ,: new . cursus )
INSERT INTO Student_lib( NCE , Nb_borrow,ST_FNAME, ST_LNAME )
VALUE( : new . NCE , : new. Nb_borrow, : new. ST_FNAME,: new. ST_LNAME )
ELSIF : new .Institution :="FST" THEN
```

```
INSERT INTO Etudiant_FST
(NCE,ST_FNAME, ST_LNAME, ADRESS, Institution
    "CLASS", CURSUS) VALUE
( : new . NCE , : new. ST_FNAME,: new. ST_LNAME, : new . adress ,
 new . Institution,: new . "CLASS" ,: new . cursus )
INSERT INTO Student_lib
( NCE , Nb_borrow,ST_FNAME, ST_LNAME )
VALUE( : new . NCE , : new. Nb_borrow, : new. ST_FNAME,: new. ST_LNAME )
ELSIF : new .Institution :="FESMT" THEN
INSERT INTO Etudiant_FESMT(NCE,ST_FNAME, ST_LNAME, ADRESS, Institution
    "CLASS", CURSUS) VALUE
( : new . NCE , : new. ST_FNAME,: new. ST_LNAME, : new . adress ,
 new . Institution,: new . "CLASS" ,: new . cursus )
INSERT INTO Student_lib( NCE , Nb_borrow,ST_FNAME, ST_LNAME )
VALUE( : new . NCE , : new. Nb_borrow, : new. ST_FNAME,: new. ST_LNAME )
ELSE
RETURN 'The university name is invalid'
END IF
END IF
when excep then
raise_application_error (−20009 , 'constraint violation' )
END;
```

To ensure tables update, a reusable procedure may be implemented to handle different table synonyms and attributes.

Table 4. A reusable Oracle update procedure

Oracle generic procedure

CREATE OR REPLACE Procedure update_table (
tbl VARCHAR2,
column VARCHAR2,
value VARCHAR2,
id__column_name VARCHAR2,
id VARCHAR2)
IS
stmt_str VARCHAR2(500);
BEGIN
stmt_str := 'UPDATE ' ‖ tble ‖ '
set ' ‖ column ‖ ' = ' ‖
valeur ‖ ' WHERE ' ‖
id_name ‖ ' = ' ‖ id ;
EXECUTE IMMEDIATE stmt_str ;
END;

This update procedure creates and executes any update on any table where column to update type is VARCHAR2. More specific tuning is to add if the procedure is intended to any column type.

5 MOTIVATION

As described previously, DDB designers are still facing the following issues:

1. DDB design is not an easy task. Multiple criteria must be considered on this sensitive operation: Sites number, exhausting user needs and frequent queries. The designer must establish a compromise between data duplication and performance's cost of update and select queries. He must find out relationship to fragmentor duplicate and the update type to consider on each synchronous or asynchronous relationship.
2. DDB implementation is still a heavy task nowadays especially with huge databases called from numerous sites. Actually this implementation, executed manually by designer, have to make a DDB look like a centralize BD, by ensuring a transparency list. This is not yet affordable automatically on current DDBMS.

The complexity level of the design of a DDB is strongly coupled to the start schema size. In production contexts, it is very common to face very complex schema with hundreds of tables. If the distribution is processed manually through Oracle command line tools, a considerable error probability is expected.
In the following, we propose, a revision of Oracle DDBMS. The idea is based on extending it by an assistance layer that provides: 1) creation of different types of fragmentation through a GUI for defining different sites geographically dispersed 2) allocation and replication of different databases. The system must automatically generate SQL scripts for each site of the original configuration.

6 New Approach proposals for Oracle DDBS

6.1 Objectives of our approach

Ideally, the new layer must satisfy the following objectives:

1. Design help for distributed schema: The layer must provide the designer a friendly and productive interface that allows him to represent the draft of the design as a comprehensive and accessible schema to review and collaboration. Fields, tables, sites suggestion lists and work tools (fragmentation and replication) must be provided to designer to ease schema graphical description and avoid additional task complication.
2. Automated implementation of design schema: Once distribution schema is established and validated by the designer along with the wizard assistance, the component "Script generator" must afford the ability to translate accurately described distribution policy to valid SQL scripts. Generated scripts can be directly executed on sites from the layer if access are already prepared, or give deliverable files to transmit to each site administrator.

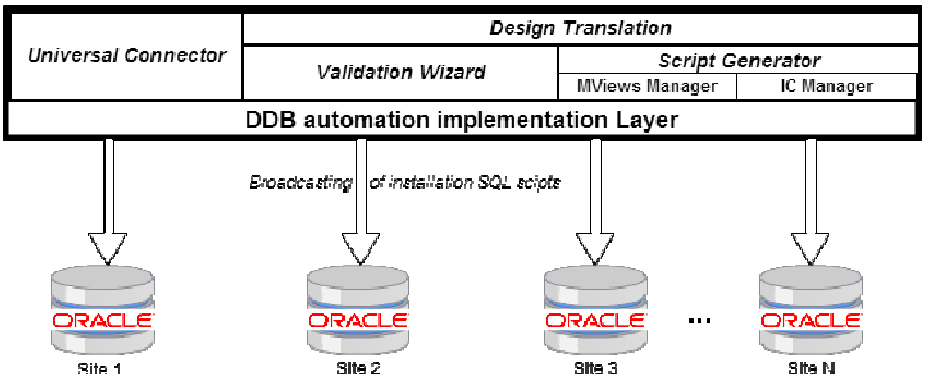
3. Ensure the integrity constraints and inter-site calculations: The fragmentation horizontal and vertical fragmentation can corrupt relations starting point for the majority of DBMS (including Oracle) do not handle the constraints of integrities between two distinct sites. It is proposed to substitute the integrity check of the local DBMS triggers that maintain the uniqueness of the primary key between sites, and the validity of the foreign keys between two or more remote fragments
4. Ensure transparency toward the end-user: Creating views that combine data from remote fragments automatically. This reconstruction is an essential constituent of the transparency of the distribution.

6.2 Suggested layer architecture:

The architecture of our Intelligent DDB, illustrated in Figure 2 depicts the architecture of the implemented layer. The assistance feature is available on the following steps of the distribution process:

1. Access to centralized database to distribute
2. DB link creation
3. Horizontal, vertical and nested fragmentation
4. Fragmentation result validation
5. Data replication

At the end of the process, two options are suggested to execute scripts, depending on afforded preconditions: 1) automatically: If design environment has a valid access to remote sites, the layer executes scripts on each remote site. 2) Manually: User transfer files using an external tool and takes in charge then execution on remote sites.



The “Universal Connector” is an abstract layer to ensure DBMS abstraction. This is the first step to heterogeneous architecture support. Through this component, the tool can access any database given relative connection string and valid credentials.
The design translation component is responsible of validating and re-writing visual description given through the GUI to a formal SQL scripts. It holds the “Validation

Wizard" that processes correctness rules on each step and reports possible failures. "Script Generator" function is dedicated to different SQL variant transcription. As a first release, only Oracle SQL variant (PLSQL) is supported. Further releases are scheduled to support T-SQL and PGPSQL variants. Inside the "Script Generator" component, "MViews Manager" ensures transparency layer by rewriting views to an inter-site transitive format. The integrity check Manager (IC Manager) generates additional procedures to ensure integrity constraints in the distributed context. In the given sample schema, the IC manager will create a specific procedure to ensure NCE unique constraint by checking the highest attributed primary key across all sites before a student subscription on the database.
The operating principle is as follows:

Table 4. Distribution processing algorithm

Algorithm 1 : Distribution Processing

BEGIN
1. Select database fragment
2. Enter the list of distribution sites
3. Retrieve the list of relationships based on initial schema
4. While " Complete Fragmentation " is false
a. Select the table fragment
b. Choose the type of fragmentation(A)
i. If Horizontal Fragmentation
1. Select column fragmentation
2. Affect the value of fragmentation for each site
ii. ELSE IF Vertical Fragmentation
1. Name the fragment
2. Select the columns of the fragment
3. Select the host site of the vertical fragment
iii. If Hybrid Fragmentation
1. Treat the hybrid fragment
c. Validate the resulting fragmentation
d. Show validation report
e. If validation is negative,return in (4.b)
f. If the resulting fragment has foreign keys
perform the derivative fragmentation
4. END WHILE
5. For each site
a. Generate scripts for creating database links from other sites
b. Generate Scripts fragments of this site
c. Generate CRUD procedures
d. Generate materialized views
e. Write the script in a file name of the site
END FOR
END

7 Intelligent-DDB

In this section we describe the resulting work and evaluate the implemented tool overall contribution and limits.

7.1 Work result

Distribution wizard "Intelligent-DDB" is intended to help users graphically distribute a centralized DB, supports the creation of DB links, horizontal, vertical, hybrid and derived fragmentation and replication. The final result is a set of SQL scripts to run on each site.

To implement our tool, we used Microsoft Windows Seven software environment. Simulation nodes in network, was made by installing two virtual machines (Oracle Virtual Box) on the chosen host. The apprehended development environment is DotNet framework 4.5 (CSharp). The figure 5 shows conduct of our assistant. Intelligent-DDB provides designers with multiple screens.

We illustrate some examples on the following:

After welcome screen and the tool introduction and interactive help access, user accesses the connection panel to identify target centralized database (Fig.3). If the designer does not want to access existing database or does not yet have the necessary access to it, a standard SDL may be written in the rich text box on the connection