Chapter 4 – Approach

4.1 Introduction

4.1.1 Purpose

In this chapter we describe the design and architectural aspects of the Database Integration System. The design is expressed in sufficient detail so as to enable all the developers to understand the underlying architecture of the system.

Highlights of the chapter:

1. Overall architecture of the system.
2. Data Design.
3. Component and interface Design.

4.1.2 Design Considerations and Guide lines

ETL (Extract – Transform – Load) and schema federation (Figure 4.1) are the underlying strategies that we have considered during the designing of the Database Integration Tool.

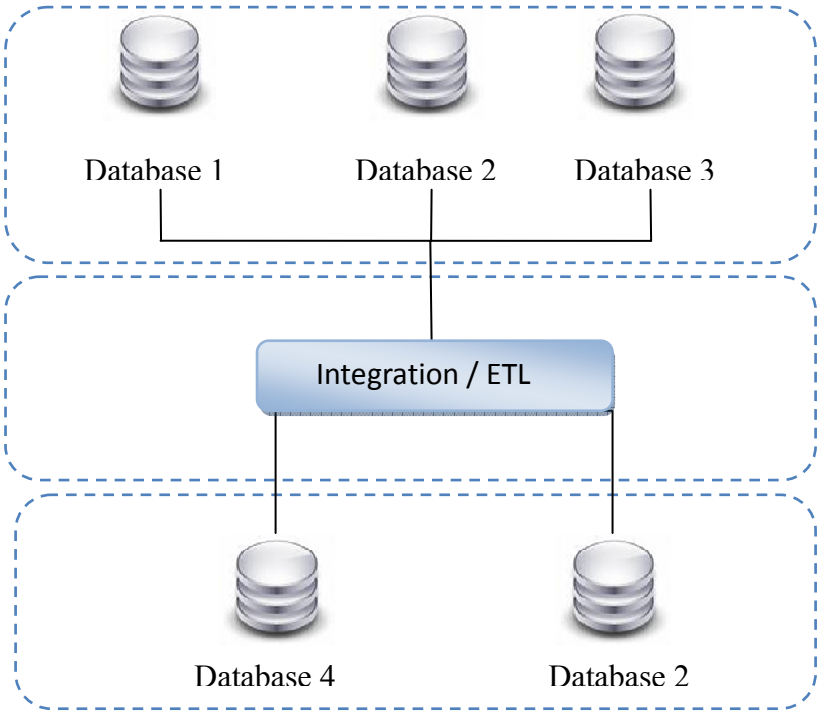


Figure 4.1 – Schema Federation and ETL

Data extracted directly from the connected databases or by combining schemas of different databases. And output is saved internally. Saved data is further manipulated and transferred to another connected database.

Extract-Transform-Load (**ETL**) is a practice that is used to take information from one or more sources, normalize it in some way to some convenient schema, and then insert it into some other repository.ETL for data warehousing, where regular updates from one or more systems are merged and refined so that analysis can be done using more specialized tools. Typically the same process is run over and over again, as new data appears in the source application(s).

All configuration files used in the Database Integration System are XML files. E.g. Connected Database configuration file, federated data description file.

Database Integration System will be developed using java and java related technologies. So all design done to be compliant with java version 6.

4.1.3 Assumptions and Dependencies

Database Integration system is platform independent and requires resources that will be depending on the size of the dataset on which the system works on.

4.2 Development Methods

Database Integration System is developed as an open source product. So by nature it is intended to expand. Object Oriented Methodology is an ideal approach for such system, because of the inherent attributes described below.

a) *Maintainable*

Object Oriented (OO) methods make code more maintainable. Identifying the source of errors becomes easier because objects are independent. The principles of good OO design contribute to an application's maintainability.

b) Reusable

Because objects contain both data and functions that operate on data, objects can be thought of as self-contained "boxes". This feature makes it easy to reuse the code in new systems. Messages provide a predefined interface to an object's data and functionality.

c) Scalable

OO applications are more scalable than their structured programming roots. As an object's interface provides a roadmap for reusing the object in new software, it also provides you with all the information you need to replace the object without affecting other code.

d) Real-World Modeling

Object-oriented system tends to model the real world in a more complete fashion than other methods. Objects are organized into classes of objects, and objects are associated with behaviors. The model is based on objects, rather than on data and processing.

e) Improved Reliability and Flexibility

Object-oriented systems are more reliable than traditional systems. Because objects can be dynamically called and accessed, new objects may be created at any time. The new objects may inherit data attributes from one, or many other objects. Behaviors may be inherited from super-classes, and novel behaviors may be added without effecting existing systems functions.

Components of the system are separated in to logical layers. Therefore Database Integration System design is strengthen by the properties of the Layered architecture such as,

1. Interoperability and Greater Compatibility with Different Databases.
2. Better Flexibility
3. Increased Life Expectancy - Increased product working life expectancies as backwards compatibility is made considerably easier.
4. Value Added Features - It is far easier to incorporate and implement value added features into products
5. Modularity
6. Task Segmentation - Breaking a large complex system into smaller more manageable subcomponents allows for easier development and implementation

4.3 Database Integration System Layered Architecture

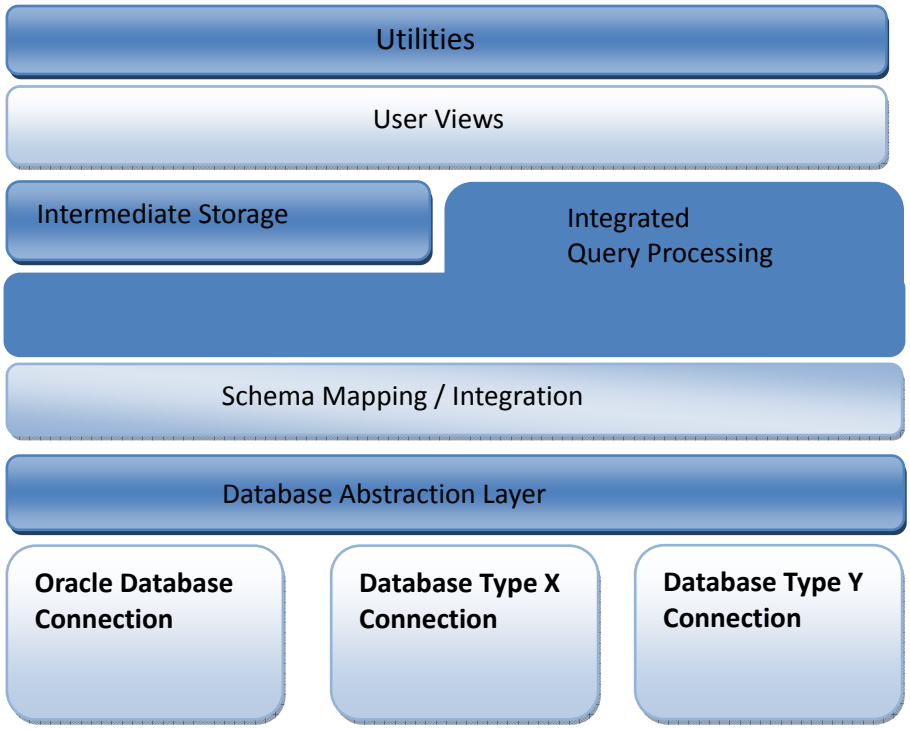


Figure 4.2 – Database Integration Tool Layered Architecture

Components of the Database Integration System are categorized in to seven layers.

1. Database Communication Layer
2. Data abstraction Layer
3. Schema mapping and Database integration Layer
4. Integrated Query Processing Layer
5. Intermediate Storage Layer.
6. User View.
7. Utilities

4.3.1 Database Communication Layer

The database communication layer consists of features to communicate with different databases. To enable communication between system and a particular database,

developers must implements an abstract interface [1]using database management system dependent Java Database Connectivity Driver[2]. The interface consists of functions to retrieve data, Meta data from databases.

4.3.2 Database Abstraction Layer

In this layer database tables and table columns are abstracted in to a common representation. Therefore the user doesn't see any difference between the storage structures of each database.

4.3.3 Schema mapping and integration Layer

Federated views are created using connected databases. And user defined views are saved in XML files. Federated data description file contain all information required to integrate databases, for an example Database reference names, selected columns, Join attributes from each table.

4.3.4 Integrated Query Processing Layer

Data retrieved from different connected databases are merged according to the federated query description.

In database federation, record set to be retrieved from each database is stored in the Federated data description file. After retrieving data sets system it-self integrate them to derive final integration view.

4.3.4 Intermediate Storage Layer

The results derived from Query processing are saved in an embedded database (derby). Saving data in an embedded database allow manipulating easily.

---

[1] An **interface** in the Java programming language is an abstract type that is used to specify an interface (in the generic sense of the term) that classes must implement.

[2] A JDBC driver is a software component enabling a Java application to interact with a database. To connect with individual databases, JDBC (the Java Database Connectivity API) requires drivers for each database. The JDBC driver gives out the connection to the database and implements the protocol for transferring the query and result between client and database.

4.3.4 User views

This layer includes functions related to query embedded database as well as connected databases.

4.3.5 Utilities

Data compression and other utilities are included in this layer.

4.4 Database Integration System - Detailed Design

In this section we describe the conceptual modeling of each layer using UML class diagrams

4.4.1 Static Modeling

4.4.1.1 Database Communication

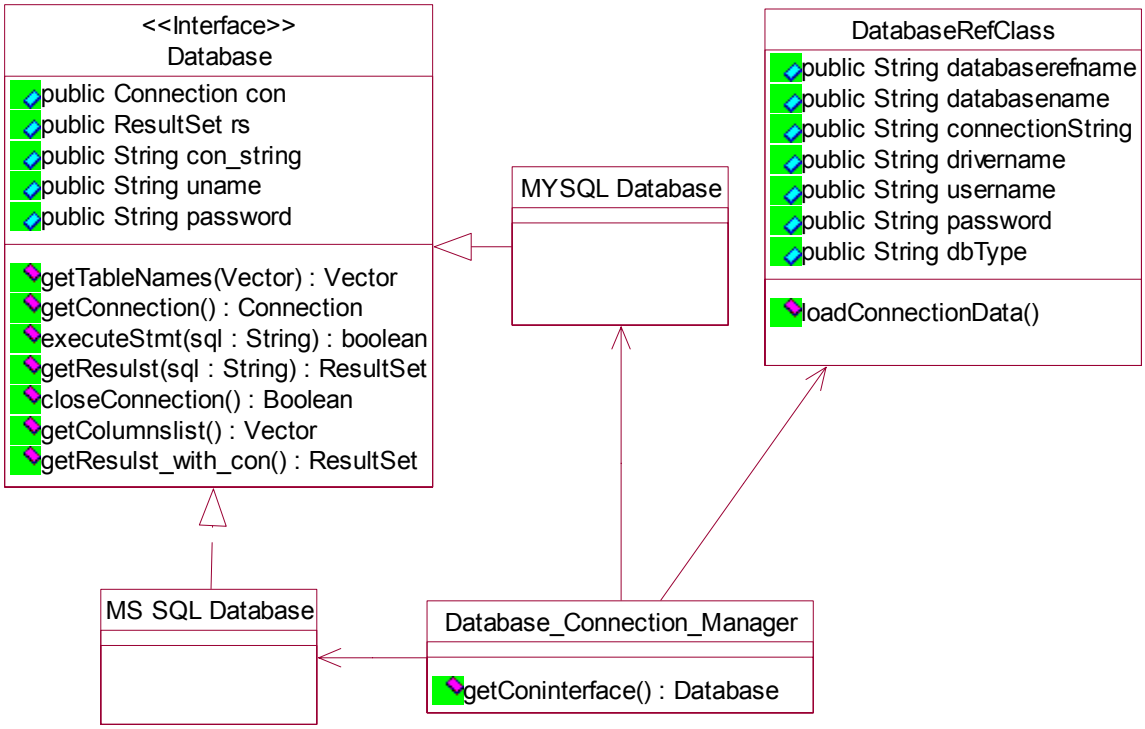


Figure 4.3 – Database Communication Classes

If user wants to connect a database to the tool, “Database” is the interface (Figure 4.3) he should be implementing using a Java Database Connectivity Driver. DatabaseRefClass contains required data to establish a connection. Each time a new connection is established, connection data is stored in a configuration file under a reference name. When a new connection is required communicating with a connected database,

connection parameters are extracted from that property file and loads to the "DatabaseRefClass". According to the properties of the "DatabaseRefClass", Database_Connection_Manager will instantiate a subclass of "Database" class. Database connection properties are saved in XML format.

Class description

Class name	Database	
Purpose	Provides an abstract interface for Database Connection	
Methods	**Method Name**	**Description**
	getTableNames(String schema)	Retrieve table names of the schema
	getResulst_with_con(String sql)	Create a connection and get Result Set
	executeStmt(String sql)throws Exception	Execute a given SQL statement
	getConnection()	Establish a connection according to a given connection String
	getColumnslist(String table)	Get Tables and it's columns
	getResulst(String sql)	Get the results from already established connection
	closeConnection()	Close open connection
Attributes	**Attribute Name**	**Description**
	con	Java.sql.Connection Object
	Con_string	Connection String
	Uname	User name
	Password	Encrypted Password
Algorithms		

Class name	DatabaseRefClass	
Purpose	Data required to make a Database connection is stored in this class	
Methods	**Method Name**	**Description**
	loadConnectionData()	Extract required Data from XML configuration file
Attributes	**Attribute Name**	**Description**
	Databaserefname	Database reference name as user defined
	Databasename	Actual instance name
	connectionString	Connection String
	Drivername	Driver to be used
	Username	User name
	Password	Encrypted password
	dbType	Database type
Algorithms	XML parser is used to extract data from configuration file using reference name.	

Class name	Database_Connection_Manager
Purpose	This class decides which subclass should be implemented according to the given data.
Methods	**Method Name** **Description** getInstence()
Attributes	**Attribute Name** **Description**
Algorithms	According to the dbtype field a subclass of Database is selected

4.4.1.2 Data Abstraction Layer

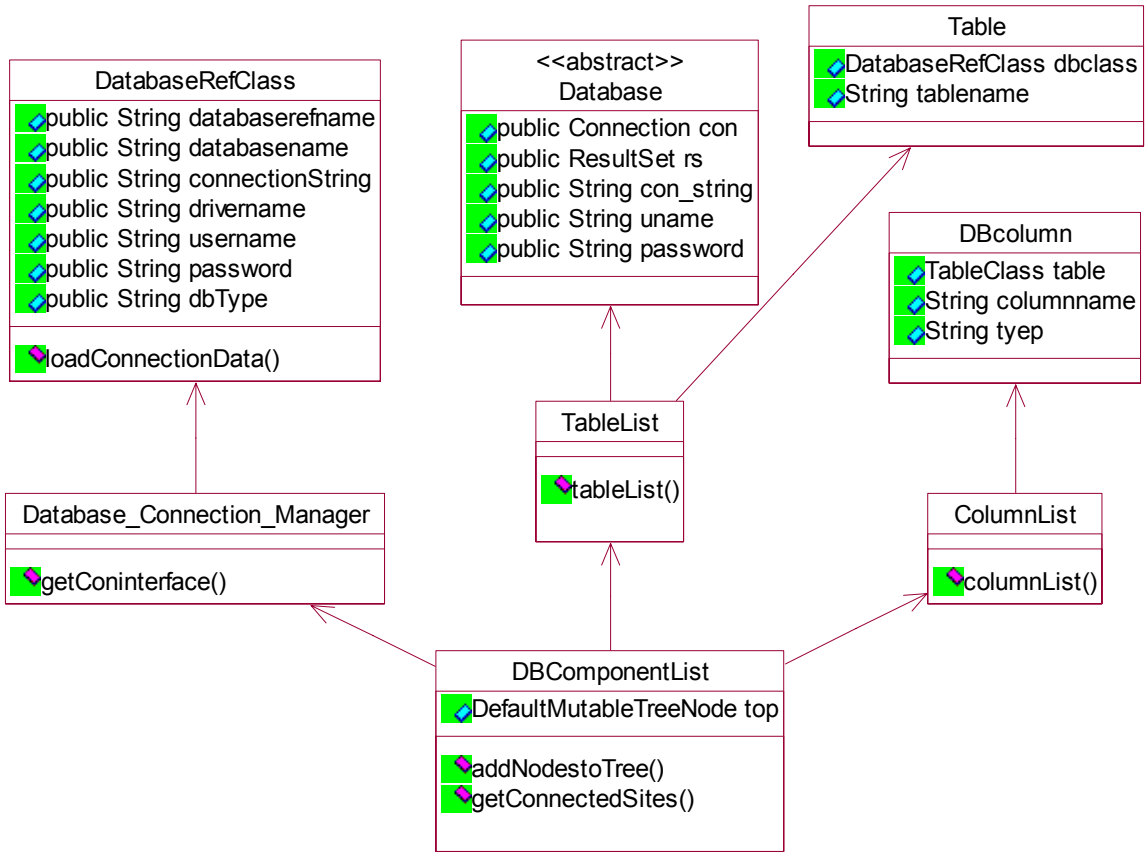


Figure 4.4 – Database independent view

Data abstraction layer builds a common view of connected databases. Tables and columns of the connected databases are organized in to a tree view without considering the underlying database type.

DATABASE A
    TABLE 1
        COLUMN 1
        COLUMN 2
    TABLE 2
        COLUMN 1
        COLUMN 2
DATABASE B
    TABLE 1
        COLUMN 1
        COLUMN 2

Tables and columns are referenced using their path names. For an example column 1 of table 1 resides in Database A is referred to DATABASE A: Table 1: Column 1

So each tables and columns can be referenced without conflicting names.

Class Description

Class name	DBComponentList	
Purpose	Graphical user interface displays a tree view listing of tables and columns of those tables. DBComponentList class associate with TableList and Column list to build table tree of connected databases	
Methods	**Method Name**	**Description**
	getConnectedSites()	Retrieve information from configuration file lists connected databases
	addNodesTotree()	Nodes add to recursively to the tree view
Attributes	**Attribute Name**	**Description**
	DefecultTreeNode	Root node for tree view
Algorithms	Connected databases are listed from the XML configuration file and each database table and its columns are added to the tree recursively. Root node is named “DATABASES” and second level nodes are reference names and third level is tables. Forth level of the tree is columns of each table.	

Class name	TableList	
Purpose	List the Tables of the connected database	
Methods	**Method Name**	**Description**
	tableList(String ref)	List tables of the connected database
Attributes	**Attribute Name**	**Description**
Algorithms	Table type classes created representing Tables of the database. Table object contains database reference name and table name	

Class name	ColumnList	
Purpose	List the columns of the connected table	
Methods	**Method Name**	**Description**
	columnList(String ref)	List columns and data type of the given table
Attributes	**Attribute Name**	**Description**
Algorithms	Given table is described; Column object is created according to the description.	

4.4. 1.3 Schema Mapping and Database Integration Layer

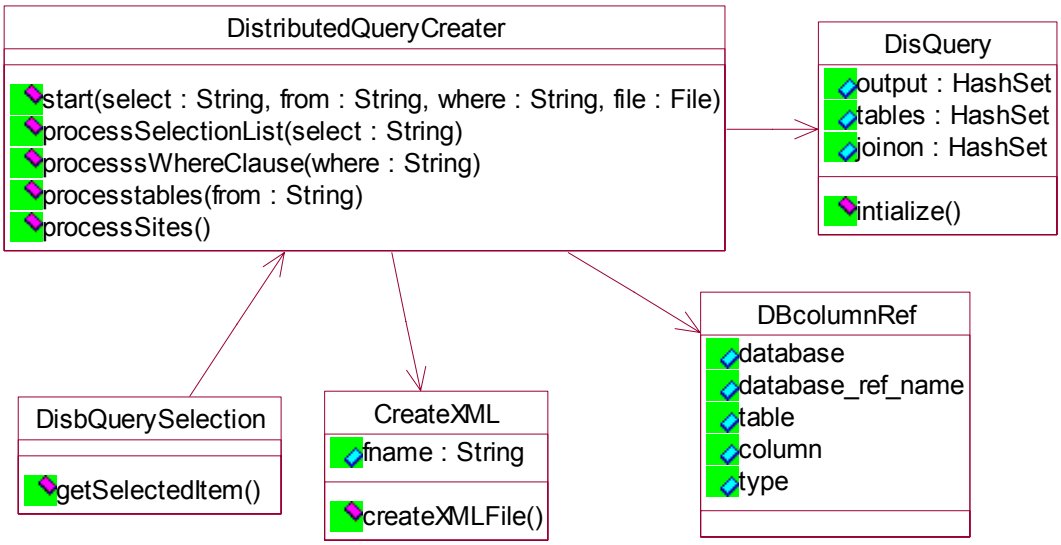


Figure 4.5 – Database Integration Layer.

Schema mapping and Integration layer primarily focusing on creating federated database views. Federated Database queries are stored in a XML file. Query processing layer interpret the file and fetch data from connected databases according to the description.

Federated data description XML file is self descriptive. It contains information about the database systems that should refer to fetch data, datasets that should retrieve from each database, what columns of the dataset must be merge to join tables.

To create a federated dataset, user selects required columns from connected databases using tree view, and tables that the selected columns belongs to. Then user should specify the join columns of the where clause just as tables resides in a one database. After that distributed query creator divides the whole query to sub queries that are suitable to execute on individual databases.

When distributed query is processing, each sub query is executed on the particular database and result dataset is extracted. In the next stage those results sets are merged according to the specified join/merge column.

Example

User generated query

Select - > database1:table1:column1, database1:table1:column2, database1:table2:column1, database2:table1:column1
From - >database1:table1, database1:table2, database2:table1
Join - > database1:table1:column1 = database2:table1:column1

DistributedQueryCreator will create the sub queries such as:
Query for database 1

Select table1 .column1, table2. Column 2
From table1, table2

Query for database 2

Select table1.column1
From table1

Class description

Class name	DistributedQueryCreator	
Purpose	Main class responsible for creating federated query description. This class process the user created query to separate sub queries that can execute on connected databases	
Methods	**Method Name**	**Description**
	Start()	Initialize Parameters
	processSelectionList()	Divide selected columns in to sub-queries
	processWhereClause()	Where clause is created for the sub queries
	processTables()	Identify the different tables of the same site and join them to create one SQL sub query
	processSites()	Identify the distinct databases

Attributes	**Attribute Name** **Description**
Algorithms	User selects the required columns from connected tables without considering the database. When processing that view tool should divide it in to sub queries. Those sub queries should be able to execute individually at connected databases. And result sets are merged by the system to produce the required integrated view.

Class name	DisbQuerySelection
Purpose	Selected object from the database tree is processed and inserted appropriately. For an example when selecting columns for "select" clause it should be in (refname:table:column) format, and table selection should be in (refname:table). Multiple tables and columns must be separated by a coma.
Methods	**Method Name** **Description** getSelectedItem() Format selected object according to the target query regain.
Attributes	**Attribute Name** **Description**
Algorithms	

Class name	CreateXML
Purpose	Federated data description is saved in XML format. Those XML files are input to the query processing layer which does the real time execution.
Methods	**Method Name** **Description**
Attributes	**Attribute Name** **Description**
Algorithms	Sample XML file Refer to appendix D

Class name	DisQuery
Purpose	Processed federated data description object representation
Methods	**Method Name** **Description**
Attributes	**Attribute Name** **Description** Output Columns of the out-put Tables Tables participated from each database joinon Join columns of the each sub-queries
Algorithms	

4.4.1.4 Integrated query processing layer

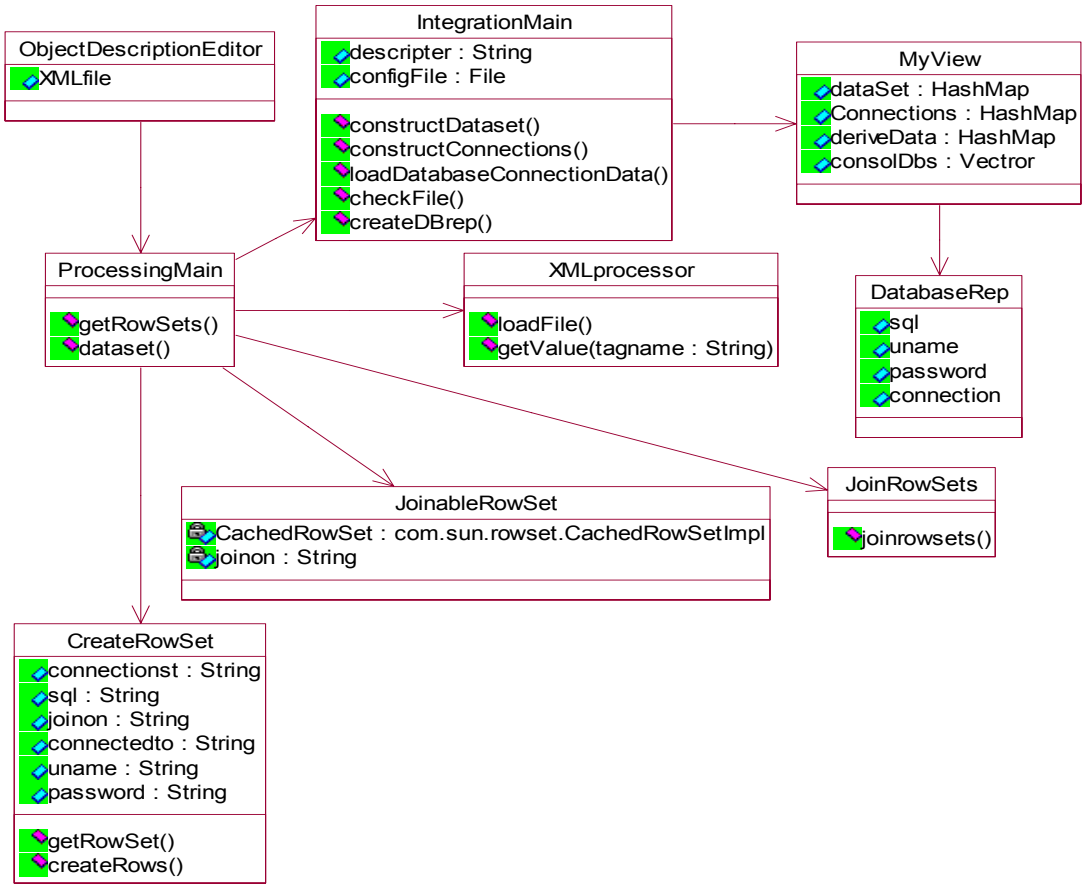


Figure 4.6 – Query Processing

Database integration layer decompose federated view in to sub-queries that each database should execute individually. As explained earlier basic terminology is execute sub queries in connected databases and result data sets are merged to generate federated view.

ObjectDeescriptionEditor is the user interface that reads the user specified configuration parameter file. Using extracted information an instance of "MYView" will be created. XMLprocessor instance will be used to extract information from the configuration file. Using "MyView" instance "IntegrationMain" class will create statement to be executed at each connection.

CreateRowSet represents a result set of a sub-query and JoinableRowSet represents a

result set and column should be used to join with other data sets. JoinableRowset contains CachRowSet and join attribute.

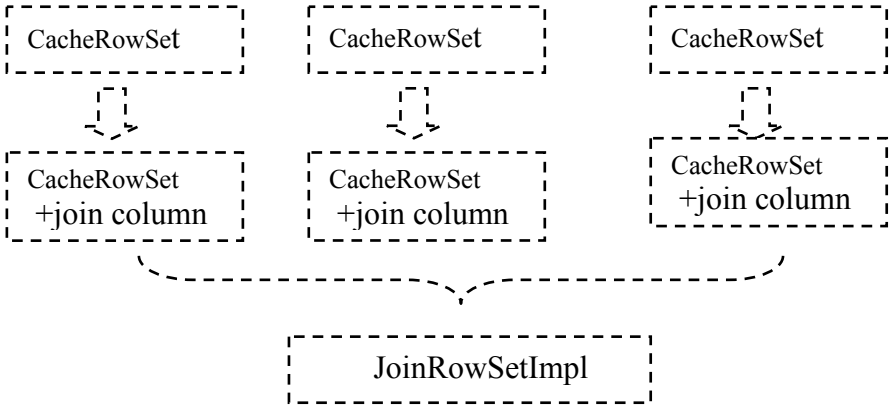


Figure 4.7 – Join Result sets

Classes and responsibilities

Class name	ProcessingMain	
Purpose	This is the control class of the database integration process	
Methods	**Method Name**	**Description**
	getRowSets()	Execute sub queries on its respective database and collect record sets
	dataset()	Join record sets collected above and create the federated view
Attributes	**Attribute Name**	**Description**
Algorithms		

Class name	IntegrationMain	
Purpose	Construct required parameters for each sub query and execute.	
Methods	**Method Name**	**Description**
	constructDataset()	Execute sub queries on connected databases
	constructConnections()	Connections of the DatabaseRep instances are established.
	loadDatabaseConnectionData()	Parse configuration file to grab the data
	checkFile()	Validate configuration file
	createDBrep()	Create DatabaseRep instances to represent database connection
Attributes	**Attribute Name**	**Description**
Algorithms		

Class name	XMLprocessor	
Purpose	XML processor basically dealing with reading attributes from the federated data description	
Methods	**Method Name**	**Description**
	loadFile()	Open and parse the configuration file
	getValue(target)	Read the value of target XML tag
Attributes	**Attribute Name**	**Description**
Algorithms	Java JDOM or any XML package bundled with java can be used to read the configuration file	

Class name	CreateRowSet	
Purpose	javax.sql.rowset. CachedRowSet is used to join different result sets upon a common field. "CreateRowSet" create a CachedRowSet for each database described in federated data configuration file.	
Methods	**Method Name**	**Description**
	createRows()	Create CachedRowSet instance
	getRowSet()	Returns JoinableRowset using output of the above method and join column described in the configuration file.
Attributes	**Attribute Name**	**Description**
	Connectionst	Connection String that state the database name and the instance name.
	Sql	SQL query that used to retrieve data
	Joinon	Join attribute that used to combine with CachedRowSet to create JoinableRowset
	Connectedto	Database reference name
	Uname	User name for login to the database
	password	Password to login to the database
Algorithms		

Class name	JoinableRowSet	
Purpose	com.sun.rowset.JoinRowSetImpl is used to join CachedRowSet's. Therefore CachedRowSet should be incorporated with join column, instance containing CachedRowSet + join column is represented by JoinableRowSet	
Methods	**Method Name**	**Description**
Attributes	**Attribute Name**	**Description**
Algorithms		

Class name	JoinRowSets
Purpose	com.sun.rowset.JoinRowSetImpl is used to join “JoinableRowSet” instances.
Methods	**Method Name** **Description** Joinrowsets() Merge JoinableRowSet’s
Attributes	**Attribute Name** **Description**
Algorithms	JoinRowSet join = new JoinRowSetImpl(); join.addRowSet(“JoinableRowSet1”,"joinOn1"); join.addRowSet(“JoinableRowSet2”," joinOn2");

Class name	MyView
Purpose	Represents the federated data object.
Methods	**Method Name** **Description**
Attributes	**Attribute Name** **Description** **Dataset** Columns of the output **Connections** Reference names of the database connections
Algorithms	

4.4.1.5 Intermediate Storage Layer

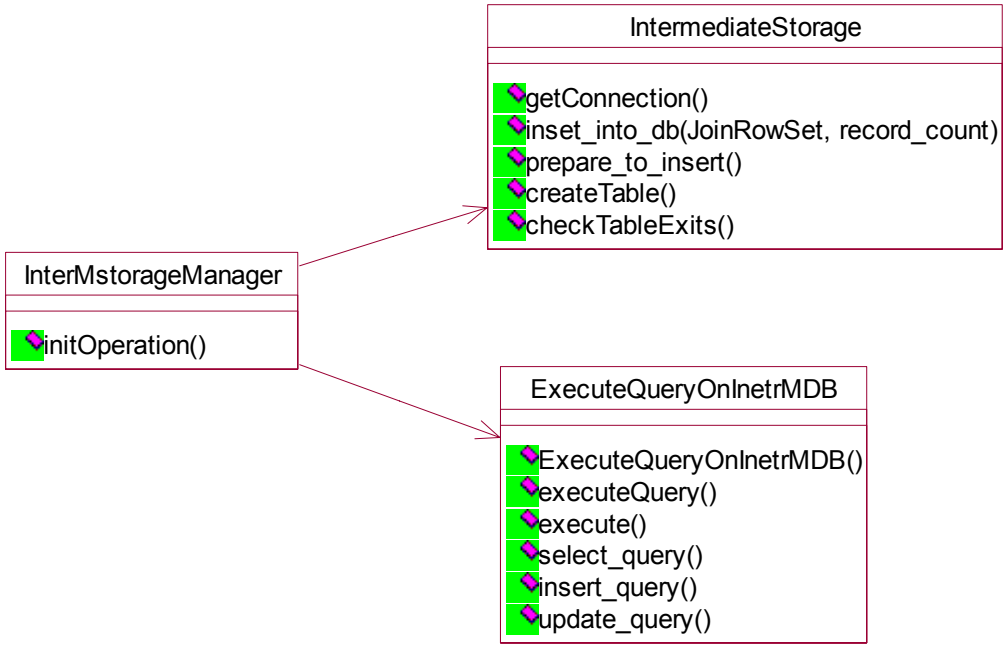


Figure 4.8 – Store data into embedded database

In transform phase of the ETL[3], extracted data is stored intermediately. And there we need apply several operations like grouping, summation and filtering to transform data. So ultimate requirement more close to consider data stored as another database. Therefore embedded database is used to store data intermediately. Derby embedded database is selected as intermediate storage.

Apache Derby, an Apache DB subproject, is an open source relational database implemented entirely in Java and available under the Apache License, Version 2.0. Apache Derby has inherent attributes that exactly match Database Integration System Requirements.

- Derby has a small footprint -- about 2 megabytes for the base engine and embedded JDBC driver.
- Derby is based on the Java, JDBC, and SQL standards.

[3] Extract Transform Load

- Derby provides an embedded JDBC driver that lets you embed Derby in any Java-based solution.
- Derby also supports the more familiar client/server mode with the Derby Network Client JDBC driver and Derby Network Server.
- Derby is easy to install, deploy, and use.
- When an application accesses a Derby database using the Embedded Derby JDBC driver, the Derby engine does not run in a separate process, and there are no separate database processes to start up and shut down. Instead, the Derby database engine runs inside the same Java Virtual Machine (JVM) as the application. So, Derby becomes part of the application just like any other jar file that the application uses.

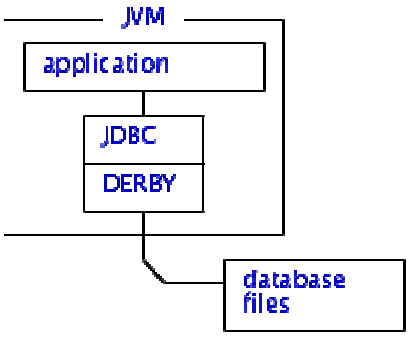


Figure 4.10 – Embedded Database Connectivity

Intermediate storage class hierarchy consists of three main classes. "InterMstoreManager" in the initial class that accepts user request and directed according to the request type. "IntermediateStorage" class basically dealing with connection management and "ExecuteQueryOnInterMDB" is for executing queries on derby database.

4.4.1.6 Data archiving using XML

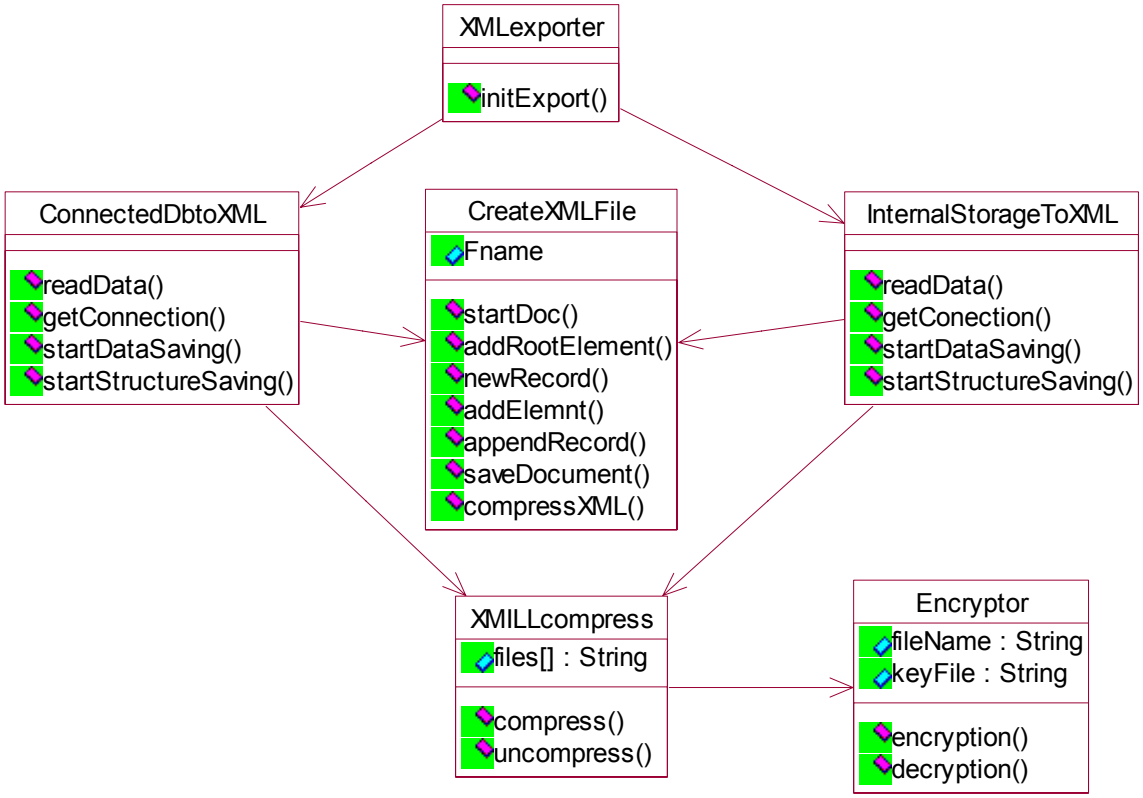


Figure 4.11 – Data export classes

Archive data using XML is another main function product expected to provide. Required data to be archived is queried from intermediate storage or connected database using a SQL statement. Retrieved data and its structure are saved in XML and compressed using XMILL. In other words archived dataset has two files, one XML file containing data and another containing structure of the dataset.

XMill is a new tool for compressing XML data efficiently. It is based on a regrouping strategy that leverages the effect of highly-efficient compression techniques in compressors such as gzip. XMill groups XML text strings with respect to their meaning and exploits similarities between those text strings for compression. Hence, XMill typically achieves much better compression rates than conventional compressors such as gzip.

XML files are typically much larger than the same data represented in some reasonably efficient domain-specific data format. One of the most intriguing results of XMill is that the conversion of proprietary data formats into XML will in fact *improve* the compression - i.e. the compressed XML file is (up to twice) smaller than the compressed original file! And this astonishing compression improvement is achieved at about the same compression speed.

Class description

Class name	XMLExporter
Purpose	Initiation class for data export. Environmental variables are loaded for the execution
Methods	**Method Name** — **Description** InitExport() — Initialize the data export.
Attributes	**Attribute Name** — **Description**
Algorithms	XMLExporter basically initiate the XML exporting process. It's initiate the required object ConnectedDBtoXML / InternalStorageToXML according to the source.

Class name	ConnectedDBTOXML / InternalStoregeToXML
Purpose	Retrieve data from the source. Initiate the XML data file, data structure file. Data is recursively add to the data file.
Methods	**Method Name** — **Description** readData() — Read the data according to the SQL query given getConnection() — Establish the connection to the source repository startDataSaving() — Recursively add records to the XML data file startStructureSaving() — Extract the retrieved dataset structure and saved in to the data structure XML file
Attributes	**Attribute Name** — **Description**
Algorithms	Dataset to be retrieved from the source is specified by SQL query. This class establishes the connection with the source and fetches the data by executing the SQL. And tool creates two files to archive the results, data file and the data-structure file. This class is responsible for creating both these files using a "CreateXMLFile" instance.

<table>
<tr><td>Class name</td><td>CreateXMLFile</td></tr>
<tr><td>Purpose</td><td>Creating XML files</td></tr>
<tr><td>Methods</td><td>
<table>
<tr><th>Method Name</th><th>Description</th></tr>
<tr><td>startDoc()</td><td>Stating a new XML document and set properties – this will create a document object in the memory</td></tr>
<tr><td>addRootElement()</td><td>Creating the root element according purpose of the XML file</td></tr>
<tr><td>newRecord()</td><td>Adding a new value to an existing element</td></tr>
<tr><td>addElement</td><td>Add new element to the file</td></tr>
<tr><td>appendRecord()</td><td>Append new value to an existing element</td></tr>
<tr><td>saveDocuemnt()</td><td>Save the document to an actual file</td></tr>
</table>
</td></tr>
<tr><td>Attributes</td><td>
<table>
<tr><th>Attribute Name</th><th>Description</th></tr>
<tr><td></td><td></td></tr>
</table>
</td></tr>
<tr><td>Algorithms</td><td>It is assumed that javax.xml package is used to create XML files. First it is supposed to create a Document object in the memory
<pre>DocumentBuilderFactory documentBuilderFactory =
DocumentBuilderFactory.newInstance();
        DocumentBuilder documentBuilder =
documentBuilderFactory.newDocumentBuilder();
        document = documentBuilder.newDocument();</pre>
and add elements as necessary
creating root element
<pre>rootElement = document.createElement(elemnt);
             document.appendChild(rootElement);</pre>
adding a new element
<pre>Element em = document.createElement(col);

      em.appendChild(document.createTextNode(val));</pre>
</td></tr>
<tr><td>Class name</td><td>XMILLCompress</td></tr>
<tr><td>Purpose</td><td>Execute XMILL to compress given list of files</td></tr>
<tr><td>Methods</td><td>
<table>
<tr><th>Method Name</th><th>Description</th></tr>
<tr><td>Compress()</td><td>compress given list of files</td></tr>
<tr><td>Uncompress()</td><td>uncompress given list of files</td></tr>
</table>
</td></tr>
<tr><td>Attributes</td><td>
<table>
<tr><th>Attribute Name</th><th>Description</th></tr>
<tr><td>Files[]</td><td>List of relative path names of files to be compressed</td></tr>
</table>
</td></tr>
<tr><td>Algorithms</td><td>XMILL binaries are used to compress given files. System will create a different process for the execution
<pre>Process process = Runtime.getRuntime().exec()</pre>
</td></tr>
</table>

XMill is a special-purpose compressor for XML documents that typically achieves substantially better compression rates. For large files, we achieved compression rates

twice as good as gzip's compression rate. XMILL works on a file-by-file basis. A given file with extension '.xml' is compressed into a file with extension '.xmi'. Any other file without extension '.xml' is compressed into a file by appending extension '.xm'. Reversely, the original file is obtained by replacing extension '.xmi' with extension '.xml' or by removing extension '.xm'.

Class name	Encryptor
Purpose	For additional security requirements compressed XML files can be encrypted. This class is used to encrypt and decrypt files. Encryption and decryption is based on an external key. Tool can generate an encryption key if user doesn't provide an acquiescent key.
Methods	**Method Name** — **Description** Encryption() — Encrypt files using a key Decryption() — decrypt files using a key
Attributes	**Attribute Name** — **Description**
Algorithms	Key size is 8192 bytes Data Encryption slandered is used for file encryption The Data Encryption Standard (DES) is a block cipher (a form of shared secret encryption) that was selected by the National Bureau of Standards as an official Federal Information Processing Standard (FIPS) for the United States in 1976 and which has subsequently enjoyed widespread use internationally.

4.4.2 Static Modeling – Package Modeling

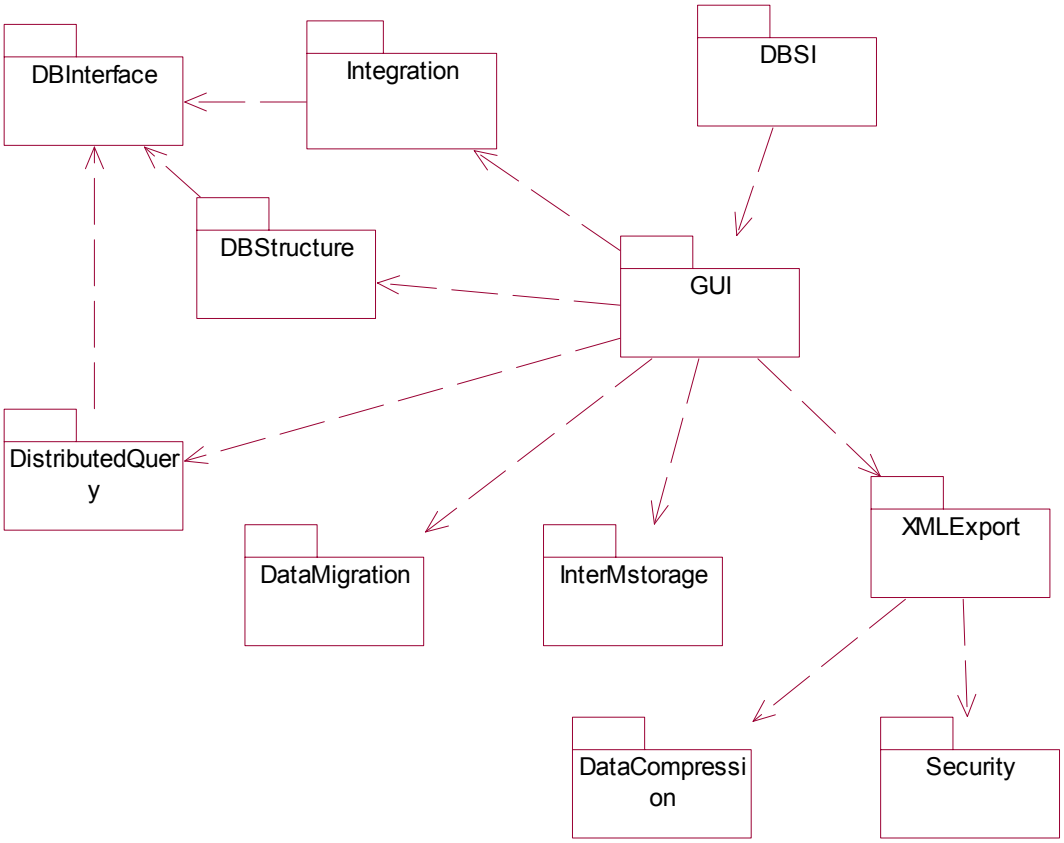


Figure 4.12 – Component Diagram

Package diagram represent the different layers of a software system to illustrate the layered architecture of a software system. DBSI is the main package initialize the program. GUI contains graphical user interface classes. “DBInterface” package contains interface classes for each database management system and “DBStructure” contains classes for generate abstract view of databases. Distributed query, Integration and InterMstorage contain classes of the corresponding layers. “XMLExport” contains class used to export data as a XML. DataCompression and Security contains classes for XMILL compression and encryption.

4.5 Dynamic Design – Object Interaction

In this Section we depicts the conceptual interaction of objects in important usage scenarios

4.5.1 Connecting to a Database

Different functions of the Database Integration Tool required establish connections with pre-configured databases. For an example query processing each sub-query has to be executed on related database.

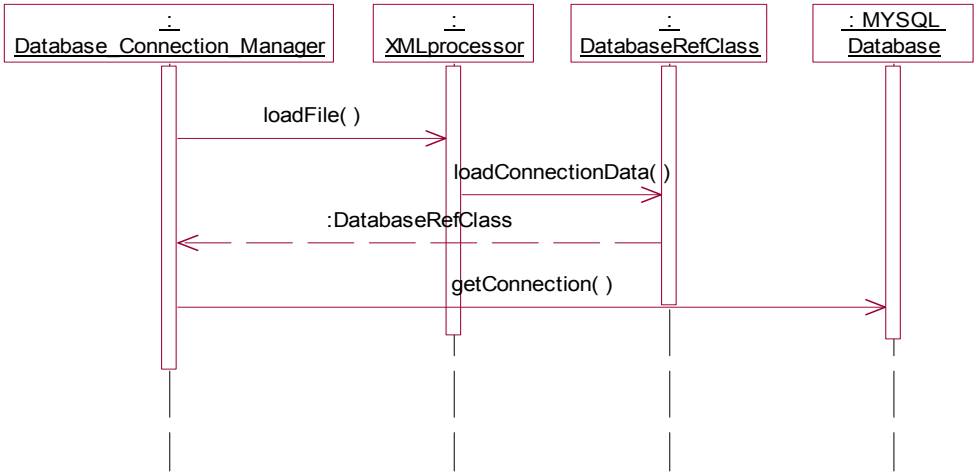


Figure 4.13 - Connecting to a Database sequence diagram

4.5.2 Add new Database reference

Before use any compliant database instance with the system, it must be added to the configuration files. New database registering function allows add new database reference.

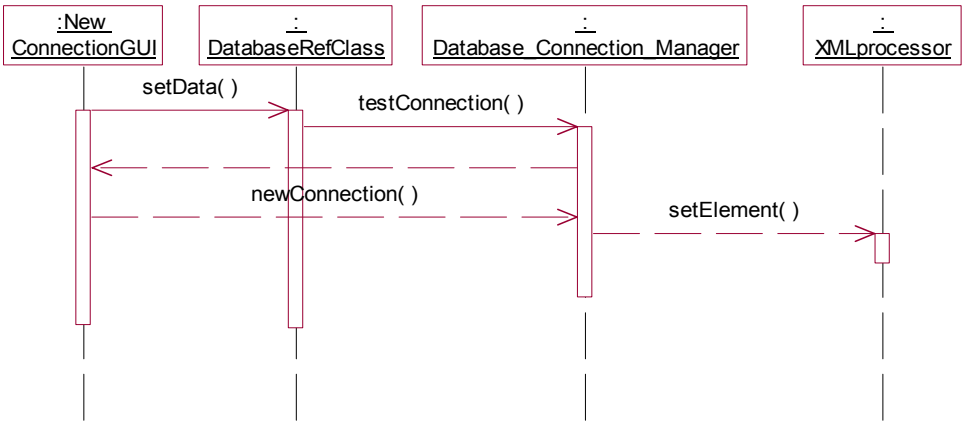


Figure 4.14 - Add new Database reference sequence diagram

4.5.3 Create Federated database view

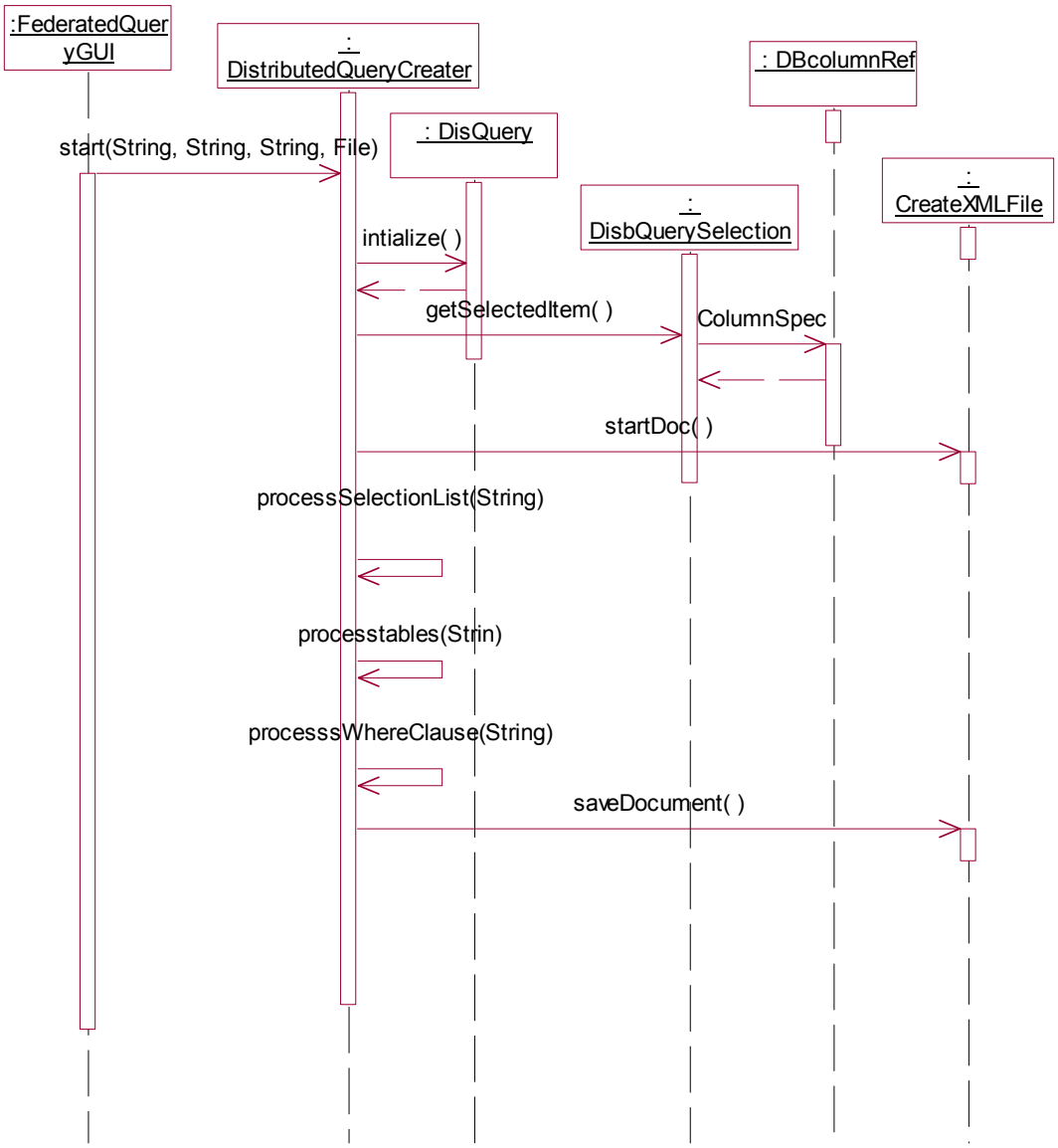


Figure 4.15 - Create Federated database view sequence diagram

"FederatedqueryGUI" is the instance of GUI that user selects columns from connected databases. "DistributedQueryCreator" subdivide the federated view in to sub queries.

4.5.4 Federated query processing

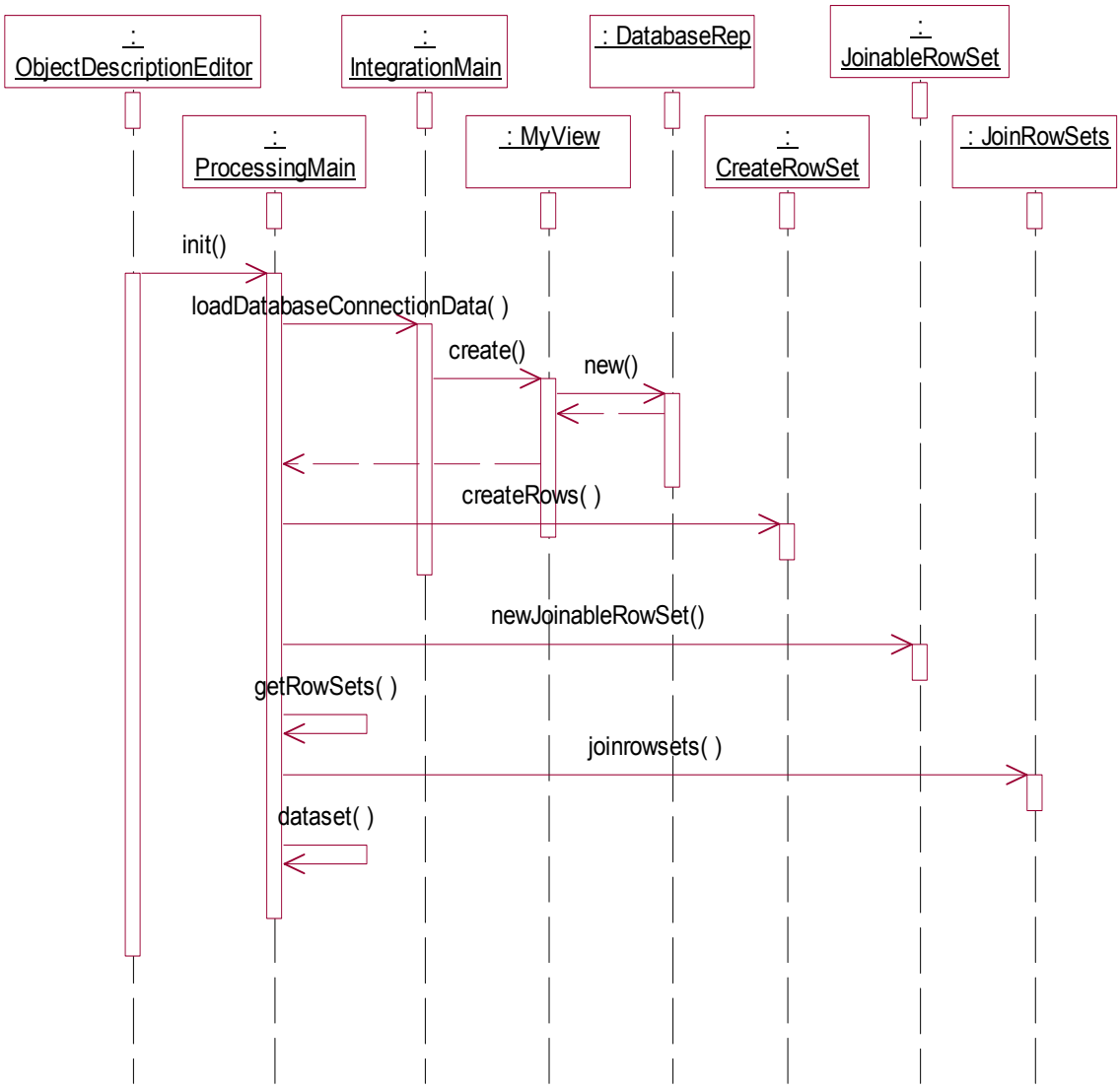


Figure 4.16 - Federated query processing sequence diagram

"ObjectDescriptionEditor" is the GUI component that loads the federated data description in to the system. And from sub-queries system creates "joinablerowset" instances. "JoinRowSet" integrates joinablerowset's to create federated view.

4.5.4 Data Archiving

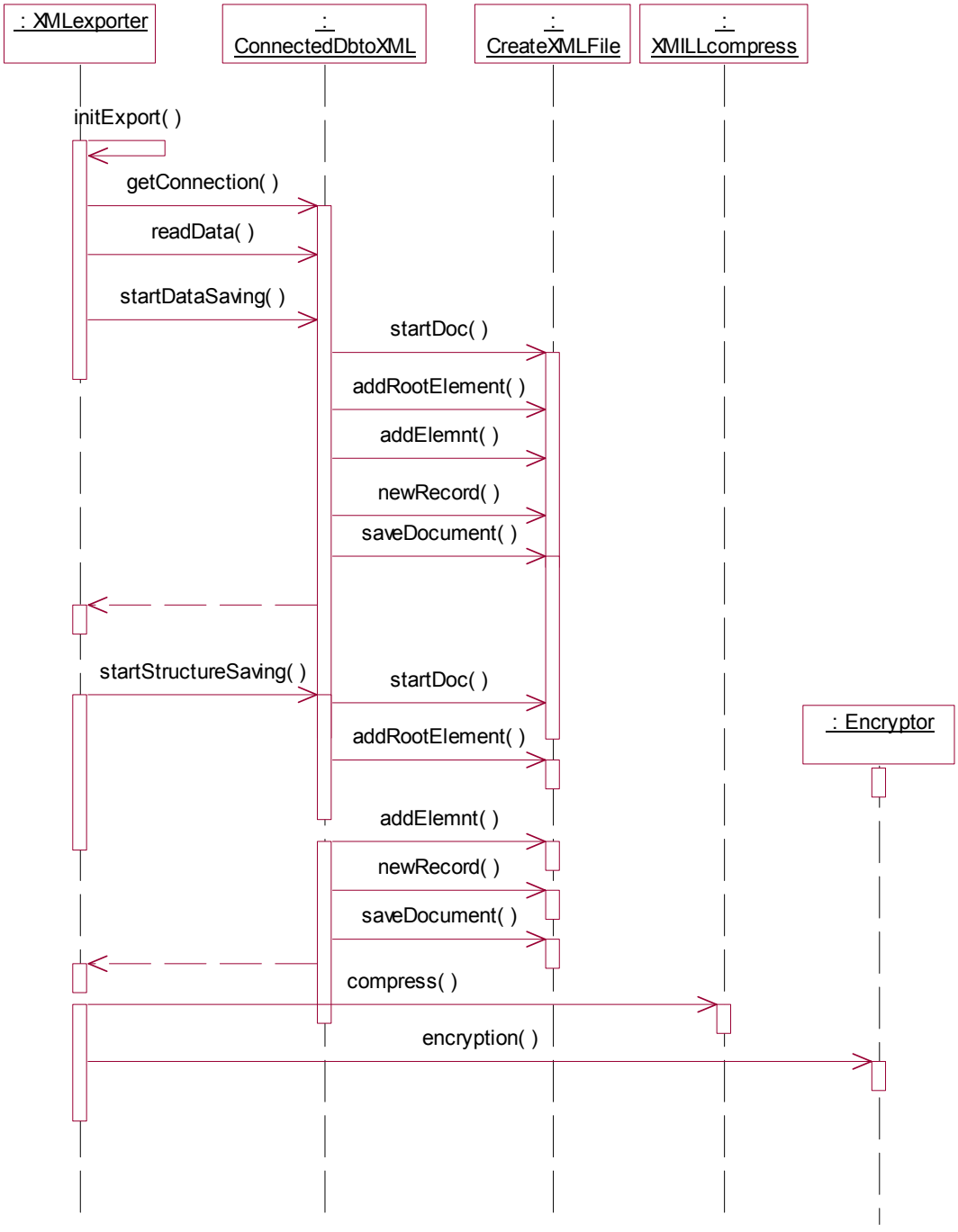


Figure 4.17 - Data Archiving sequence diagram

Saved data files are provided to the XMILLcompress as an input. If user requires additional security, compressed files can be encrypted using an “Encryptor” instance.

4.6 Graphical User Interface

When designing Graphical User Interface (GUI) for the Database Integration Tool, we mainly focused on the need of working with several database instances. Also other high lighted requirements such as comparing different data sets, structural mapping of different data sets. And users of the tool are well experienced with software products.

So IDE (Integrated Development Environment) style user interface is designed as the main interface of the Database Integration Tool.

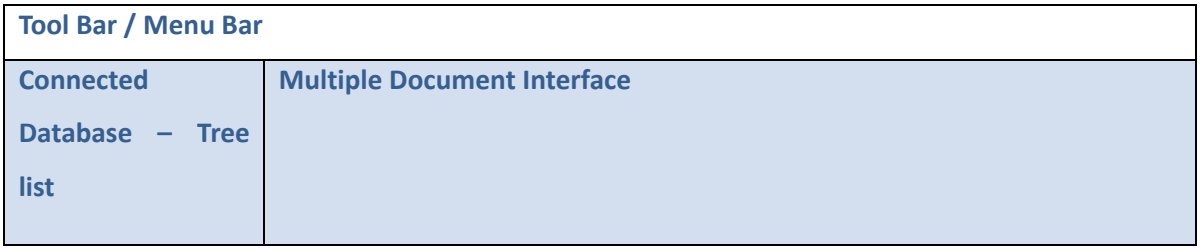


Figure 4.18 – Main Interface.

Designed GUI for the Database Integration tool is mainly consist of three parts, Menu bar, left panel consist of tree view of connected database and right side Multiple Document Interface. All functional windows are open inside MDI interface. Therefore database structure can be shared among all other functional interfaces. Other advantages of using above GUI model can be stated as below,

- With MDI, a single menu bar and/or toolbar is shared between all child windows, reducing clutter and increasing efficient use of screen space.
- An application's child windows can be hidden/shown/minimized/maximized as a whole.
- Features such as "Tile" and "Cascade" can be implemented for the child windows.
- Possibly faster and more memory efficient, since the application is shared, and only the document changes. The speed of switching between the internal windows is usually faster than having the Operating System switch between external windows.
- Can have keyboard shortcuts to quickly jump to the functionality you need (faster navigating), and this doesn't need the OS or window manager support, since it happens inside the application.

4.6.1 Graphical User Interface Detail Design

New Database Connection Window

Connection Reference name	Text field	
Connection Type	Drop down	
Database Server	Text field	
Database Instance	Text field	
User name	Text field	
Password	Text field	
	Add connection Button	**Check Connection Button**

Connection Type – list box of database types that system supports

Database Server – Server name connection manager should referenced

Database instance – instance name of the database

User name – login name

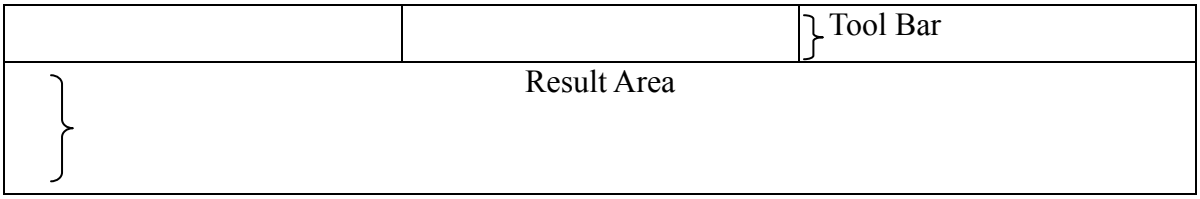
User should check the connection first. Initially Add connection button is deactivated. If only the connection is valid and possible to establish tool will activate the "Add connection button".

New Federated Data description

Selected column list	Text field	
Table List	Text field	
Join column List	Text field	
Create query Button		**Edit Existing query button**

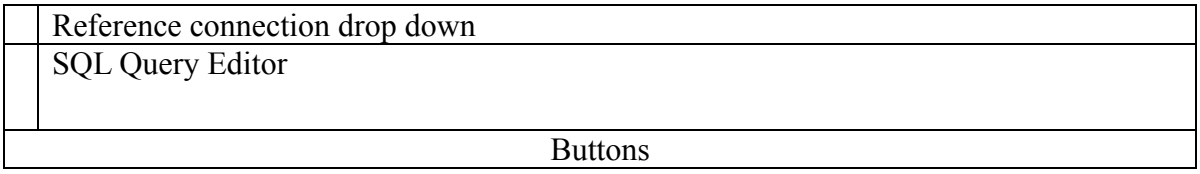
Vertical scrollbars of the above text fields has to be disabled.

Integrated Query Processing Window



Integrated query window contain main two areas, toolbar and result area. Results area shows processing status, query results of the each operation.

Data Migration Window



In data migration window user should be able to select the source database. SQL Editor specifies the query for selecting data set. Buttons contain for view result set, save result set, export result set, export XML operations. Results of a query will be displayed in a separate window. Save result set option will save the result in internal database. Export result window should allow selecting a destination