

Data Connectivity Driver(ODBC) for JIRA

Project report submitted in partial fulfillment of the requirement for the degree of
Bachelor of Technology

in

Computer Science and Engineering/Information Technology

By

Prakhar Srivastava (141297)

Under the supervision of

Mr. Srinath Gunasekaran

to



Department of Computer Science & Engineering and Information Technology

**Jaypee University of Information Technology Waknaghat, Solan-173234,
Himachal Pradesh**

Certificate

Candidate's Declaration:

I hereby declare that the work presented in this report entitled “ Data Connectivity Driver(ODBC) for JIRA” in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering/Information Technology submitted in the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat is an authentic record of my own work carried out over a period from February 2018 to May 2018 under the supervision of **Mr. Srinath Gunasekaran**, Director of Engineering, Dept. of Magnitude Software, Bangalore.

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Prakahr Srivastava

141297


This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Mr. Srinath Gunasekaran

Director of Engineering

Magnitude Software

Dated:


7/ May/ 2018

Acknowledgement

I would like to take the opportunity to thank and express my deep sense of gratitude to my mentor and project guide **Mr. Srinath Gunasekaran** for his immense support and valuable guidance without which it would not have been possible to reach at this stage of our final year project.

I am also obliged to all my faculty members for their valuable support in their respective fields which helped me in reaching at this stage of my project.

Table of Contents

Certificate.....	(i)
Acknowledgement.....	(ii)
List of Figures.....	(iv)
List of Tables.....	(v)
List of Graphs.....	(vi)
Abstract.....	(vii)
1. Chapter-1 Introduction.....	1
2. Chapter-2 Literature Survey.....	8
3. Chapter-3 System Development.....	9
4. Chapter-4 Performance Analysis.....	31
5. Chapter-5 Conclusion.....	49
References.....	51

List of Figures:

1. Figure 1: Architecture of JIRA ODBC Driver.....	20
2. Figure 2: High Level Design of the Driver.....	21
3. Figure 3: Low Level Design of the Driver.....	22
4. Figure 4: Data Flow Diagram.....	24
5. Figure 5: Structure of MDEF file.....	45
6. Figure 6: Schema of the tables.....	46
7. Figure 7. Sample Output.....	47

List of Tables:

1. Table 1: Unit Test 1.....	33
2. Table 2: Unit Test 2.....	35
3. Table 3: Unit Test 3.....	37
4. Table 4: Integration Test 1.....	39
5. Table 5: Integration Test 2.....	41
6. Table 6: System Test 1.....	43

List of Graphs:

Abstract

An ODBC driver uses the Open Database Connectivity (ODBC) interface by Microsoft that allows applications to access data in database management systems (DBMS) using SQL as a standard for accessing the data. ODBC permits maximum interoperability, which means a single application can access different DBMS. Application end users can then add ODBC database drivers to link the application to their choice of DBMS.

The ODBC solution for accessing data led to ODBC database drivers, which are dynamic-link libraries on Windows and shared objects on Linux/UNIX. These drivers allow an application to gain access to one or more data sources. ODBC provides a standard interface to allow application developers and vendors of database drivers to exchange data between applications and data sources.

CHAPTER-1

INTRODUCTION

1.1 Introduction

An ODBC driver uses the Open Database Connectivity (ODBC) interface by Microsoft that allows applications to access data in database management systems (DBMS) using SQL as a standard for accessing the data. ODBC permits maximum interoperability, which means a single application can access different DBMS. Application end users can then add ODBC database drivers to link the application to their choice of DBMS.

JIRA is a proprietary issue tracking product, developed by Atlassian. It provides bug tracking, issue tracking, and project management functions. It has been developed since 2002. According to one ranking method, as of June 2017, Jira is the most popular issue management tool.

Terminology

Terminologies related to the system have been explained in this section.

- ODBC (Open Database Connectivity):
ODBC (Open Database Connectivity) is a standard application programming interface (API) for accessing data present in the database management systems. It is independent of database systems and operating systems. It accomplishes independence by using an ODBC driver as a translation layer between application and the database.
- JIRA:
JIRA is a software developed by Atlassian which is one of the best software development tool used by Agile teams. JIRA is built for every member of a software team to plan, track and release great software.

The Jira ODBC Driver allows you to easily connect to live Jira data through Excel, Tableau, Power BI, Qlik or any analytics application. Jira stores data in structures which do not follow the rules of data typing and structure that apply to traditional relational tables and columns. Because traditional ODBC toolsets might not support these data structures, the data needs to be mapped to a relational form. The Simba Jira ODBC Driver uses schema tables to map the Jira data to an ODBC-compatible format, allowing users to report in their BI and analytics tool of choice.

1.2 Problem Statement

The designers of ODBC aimed to make it independent of database systems and operating systems. An application written using ODBC can be ported to other platforms, both on the client and server side, with few changes to the data access code.

Business users need access to data scattered across the enterprise. Time pressures and scant engineering resources make it difficult for enterprises to quickly deliver data from multiple data sources in useful formats for operational and strategic use. Magnitude's data connectivity solutions can help.

If an application consumes, creates or exposes data, it needs a flexible, powerful solution that can handle the market requirements of its current and future data connectivity needs. The Magnitude Simba connectivity drivers include a full range of data connectivity solutions based on SQL, ODBC, JDBC, ADO.NET, XMLA and MDX.

1.3 Objectives

- a. Proven Connectivity: To quickly connect any data access standard-compliant application with Jira data without costly in-house development and maintenance.
- b. Flexibility: To deploy Jira ODBC driver on a desktop, server, or as a Cloud Solution with a high-performance wire protocol serving hundreds of clients. Jira ODBC driver can also be deployed as a branded or white box solution for customers.
- c. High Compatibility: The Jira ODBC driver is rigorously tested by Simba and our partners all day to verify compatibility with supported data sources and BI tools.

The project intends to solve the problem of connecting to the non-SQL databases using SQL as standard for connecting. Below are a few applications to which the driver provides solutions:

- To execute SQL statements on JIRA database

The driver can be used to insert, delete and fetch data from the JIRA database using SQL statements.

- Enabling Business Intelligence (BI)

The driver connects to the BI tools like PowerBI, Netezza etc to enable business intelligence form the data present in the JIRA database.

1.4 Methodology

The input which is an SQL statement is parsed to identify the columns and tables and goes through two stages which is parsing metadata and fetching data from the database. In addition to these, there is another stage in which the fetched data is converted to flat tables to display. In detail, these stages are:

Step 1: Parse Metadata

The metadata of the database is in JSON format which contains the data about all the schemas, tables, columns, methods, stored procedures and the APIs to send GET, POST, POST or PUT requests. The table name parsed from input is searched from the metadata and the columns and APIs are parsed with which the request is sent through the API calls.

Step 2: Fetch data from database

The requests sent through APIs for data will be sent with a response. The response contains the data for which the request is sent. GET request is sent in order to fetch data from the database using the respective APIs, POST request is sent to add a row to a table in the database, PUT request is sent to edit some columns for the existing rows in a table in the database and DELETE request is sent to delete a row from a table in the database.

Step 3: Display data in the form of flat table.

In this stage, the objective is to convert the format of data fetched which is from schema-less to flat tables. The code parses the columns of the tables from the metadata of the database and uses that information to convert the data into flat tables and display to the users the fetched data.

1.5 Organization

The project focuses on the development of a driver which is used to access a non-SQL database using SQL as a standard. The main body of the thesis is preceded by a table of contents including a list of figures, tables and glossary, followed by the units used in the report, followed by appendices, which contain screenshots as well as part of the source code. The body of the thesis contains introduction to the project, a literature survey done at the beginning of the project to collect the requirements, followed by high-level design which focuses on developing the system architecture followed by detailed design where the system is broken down into modules. The thesis also contains details of development and deployment environment used during the implementation of the project. The organization of the rest of the thesis is as follows:

Chapter 1 is an Introduction to the system. It lists the terminology, problem statement, motivation, objectives, scope of the project, literature survey and the methodology applied in the system.

Chapter 2 discusses the summarised overview of the various research papers taken into consideration along with important facts and figures completed with the required references utilised in those research papers.

Chapter 3 is Software Requirements Specification, which explains the user characteristics, assumptions, dependencies, constraints and functional requirements. High Level Design, which explains architectural strategies, system architecture and flow of data in the system through the data flow diagrams. It also contains the detailed design, which focuses on the structure chart of this project and flow chart diagrams. It explains the key modules involved in the project as well as details regarding how they can be implemented.

Chapter 4 is Software Testing, which explains the test environment and briefly explains the test cases which were executed during unit testing, functional testing and integration testing. The entire system testing is also done and the details listed in this chapter.

Chapter 5 is the Conclusion, which gives the outcome of the project work carried out and also brings out the limitations of the project and elaborates on future enhancements to improve the system.

CHAPTER-2

LITERATURE SURVEY

Seifedine Kadry in “**An Implementation of ODBC Web Service**”, says Web applications built using .NET technologies usually access relational databases via ODBC API. This requires a database system specific ODBC driver to be installed on the application side. On the other hand, a paradigm shift is taking place in web application architectures. Future web applications will be built around Service-Oriented Architectures (SOA) where applications will be assembled using remote “web services” components. These newly assembled applications will provide functionalities using remote web services components over Internet via XML messages. Same paradigm shift will eventually apply to the data communication between applications and databases. Future applications will utilize standard “database web services” for data storage and querying requirements. Here we propose a new architectural model and present a prototype “database web service” for relational database systems. In our model, web applications do not need to deal with database drivers but leave the driver-oriented communication to a database web service. Our database web service eliminates the need for installation, maintenance, and other issues involved in maintaining ODBC drivers in distributed web application development. J. Williams in “**ODBC and its Advancements**”, says Relational databases can be accessed by general purpose using standard ODBC API. This requires applications to use a ODBC driver specifically developed for ODBC access to the database system that needs to be accessed. Therefore, wide-distribution of such an application requires the driver to be also shipped to the client. Future changes to the driver require driver updates for every Problem is even more complicated if the application needs to access many different relational database systems. Basically, this requires distribution and maintenance of many drivers in client-side. Here we propose a new approach for applications to access database systems. It is based on Web Services framework. In this approach, client is reduced to a thin client, just like web applications took the client-side computations to server side. In this case, database drivers are the problem, so they are placed on a server and moved away from the client. And, applications are provided a new interface to access database systems in a uniform way, similar to ODBC API provided a uniform access mechanism for all relational database systems.

CHAPTER-3

SYSTEM DEVELOPMENT

The purpose of this section is to present a detailed description of the Data Connectivity Driver (ODBC) for JIRA. This will include the scope and features of the system, the interfaces of the system along with what the system will do and response of the system. The chapter is intended for both the users and developers of the system. It provides an overview of the system functionality and system interaction with other systems. The methodology document describes all data, functional and behavioral requirements of the software under production or development. It consists of the sections: Overall Description and the Specific Requirements. The overall description gives a bird's eye view of the product and its perspectives, its functions, the user characteristics, general constraints and chief assumptions and dependencies. In the specific requirements section, the product specific technical constraints and requirements described. Requirements include functional requirements, performance requirements, software requirements and hardware requirements and specific technical constraints include design constraints. The environment of the functioning of the system, tools and platforms used in development and testing and other such requirements and described in this section.

3.1 Overview

This section intends to provide an overview of the entire system. The system is explained in its context to show how it interacts with other systems and introduce its basic functions. It also intends to explain different stakeholders that will use the system and functionalities available for each type. At the end, constraints and assumptions for the system are presented.

3.1.1 Product Perspective

The JIRA ODBC Driver will be able to connect data quickly for comprehensive business intelligence and enables reporting on data that is stored in JIRA. The Driver is scalable, flexible, powerful and can handle specific market requirements of current and future data connectivity needs.

3.1.2 Product Functions

The primary function of the driver is to use Open Database Connectivity (ODBC) interface that allows applications to access data in JIRA database management systems (DBMS) using SQL as a standard for accessing the data.

3.1.3 Constraints

The JIRA ODBC Driver has the following constraints

- The APIs used to get data fetches only limited rows of data for which pagination has to be enabled to fetch more data.
- There are some virtual tables which are present within tables and are not displayed directly. So, expansion has to be done on such tables to display data of those virtual tables.

3.1.4 Assumptions and Dependencies

The driver parses the metadata of the JIRA database and also the data by fetching data using the APIs. The presumptions and conditions contemplated while doing the comparative investigation for JIRA Driver are

- Metadata of the database is written in JSON format.
- The Driver complies with the ODBC 3.80 standard.

3.2 Specific Requirements

The accompanying area talks about the necessities identified with the framework's usefulness, execution and its interface. It incorporates fundamental equipment and programming required for the framework to do its assignments. This area depicts the product necessity of the venture in detail, sufficiently adequate to fulfil those prerequisites, and analysers to test the framework actualizes those necessities.

3.2.1 Functional Requirements

Usefulness of this JIRA ODBC Driver depicts in detail how the framework functions and the productivity as for every other calculation and information structure. It incorporates the contribution to be nourished to the framework, the preparing of information, and the normal yield. Handling incorporates the techniques and capacities used to work upon the information, contrast it with database and concentrate data required. At long last the yield incorporates the yield that is normal from a given info set

3.1.4 Assumptions and Dependencies

The system is involved in Caching and Metering. The presumptions and conditions contemplated while running this system are

3.2.1.1 Functional Requirement 1

ID: FR1

TITLE: Access to the application.

DESC: The client ought to have the capacity to get to the application by tapping the executable gave.

RAT: For the client to access the system.

DEP: None

3.2.1.2 Functional Requirement 2

ID: FR2

TITLE: Input.

DESC: The client writes an SQL statement.

RAT: For the client to provide the system with input.

DEP: FR1

3.2.1.3 Functional Requirement 3

ID: FR3

TITLE: Input.

DESC: The whole database in connected to the driver to perform BI.

RAT: For the client to use the system.

DEP: FR2

3.2.1.4 Functional Requirement 4

ID: FR4

TITLE: Input handling.

DESC: Given input SQL select, insert or delete statements to the driver, it provides the result of the data fetched or updated in the database.

DEP: FR3

RAT: Output for the user 3.2.1.3 **3.2.1.5**

3.2.1.5 Functional Requirement 5

ID: FR5

TITLE: Output in ODBC Test Tool.

DESC: The output of the system is provided as flat tables from the ODBC Test Tool for the SQL statement written.

RAT: Output for the client.

DEP: FR4.

3.2.2 Performance Requirements

The prerequisites in this area give an itemized detail of the client collaboration with the product and estimations put on the framework execution.

3.2.2.1 Usage of the Result in Graph / Chart format

ID: QR1

TITLE: Use of the result in graph organize.

DESC: The output of the system should easy to use and simple to utilize.

3.2.2.2 Response time

ID: QR2

TITLE: Response time

GIST: The fastness of processing the input.

SCALE: The time difference between output from and input to the system.

METER: Readings so obtained from 100 runs of the system.

MUST: No more than 1 millisecond 100% of the time.

WISH: No more than 1 millisecond 100% of the time

3.2.3 Design Constraints

The section includes the design constraints on the software caused by the hardware.

3.2.3.1 Hard Drive Space

ID: QR3

TAG: HardDriveSpace.

GIST: HardDriveSpace.

SCALE: The application need of hard drive space.

METER: GB

MUST: No more than twice the size of the Metadata file.

PLAN: No more than 2 times the size of Metadata file.

WISH: As much as possible.

3.2.3.2 Application Memory Usage

ID: QR4

TAG: ApplicationMemoryUsage.

GIST: ApplicationMemoryUsage.

SCALE: Observations during the testing.

METER: GB

MUST: No more than twice the size of the Metadata of the database.

PLAN: No more than 2 times the size of Metadata of the database.

WISH: As much as possible.

GB: DEFINED: Gigabyte

3.2.3.3 Graphical Memory Usage

ID: QR5

TAG: GraphicalMemoryUsage.

GIST: GraphicalMemoryUsage.

SCALE: Graphical requirements during testing.

METER: GB.

MUST: 4GB.

PLAN: 10-45GB

WISH: As much as possible

3.2.4 Software Requirements

The following are the vital programming devices required for the execution of JIRA ODBC Driver:

- Operating System: Windows, Linux and Mac OS X.
- RAM Size: 16GB or more
- Storage: 25GB or more
- Visual C++
- Microsoft Visual Studio

3.2.5 Hardware Requirements

The base equipment necessities that are basic to run JIRA ODBC Driver is recorded underneath: The system is hosted on the Windows server that can be remotely logged onto when the administrator needs to check up on the performance of the controller. There is no requirement for a hardware interface for this project.

3.3 Interfaces

The project needs specific user interface and a general software interface.

3.3.1 User interface

A graphical user interface has to be provided for taking input and displaying the output which is the ODBC Test Tool.

3.3.2 Hardware Interface

There is no need of a special hardware interface for this system.

3.3.3 Software interface

Since every one of the apparatuses being created is for remain solitary desktop programming, every one of the modules of the device ought to be available on the framework.

Design of JIRA ODBC Driver

One of the essential stages in the product improvement cycle is the outline stage. In this stage, framework association is created such that it fulfills both useful and non-useful necessity. In this stage, bigger framework is decayed into littler modules for better comprehension of the outline.

The portrayal of the product engineering informs you concerning the yield of the venture. A portion of the difficulties confronted amid the plan stage are outline thought, presumptions and conditions which are clarified in this part. This section talks about the abnormal state plan and nitty gritty outline of fisheries observing framework utilizing different methods. The plan stage helps in finding any sort of impediment while building up the venture, and furthermore helps in improving the execution of the system framework.

High Level Design

Design considerations, architectural strategies, system architecture, Data Flow Diagrams and Context Flow Diagrams all comes under high level design and these are explained as follows:

Design Considerations

Issues identified with a few outline contemplations must be tended to before planning a total answer for the framework. These issues incorporate design procedures and approach utilized for the advancement, worldwide restrictions or limitations, presumptions and conditions as for programming. An abnormal state diagram of various outline of usefulness contemplations is incorporated under this segment.

Assumptions and Dependencies

- The graphical interface used for the driver is ODBC Test Tool and C++ is used as the programming language
- The minimum hardware requirements that are required for implementation of this system are 4GB RAM.

General Constraints

Following constraints are kept in mind while developing the code:

- a. The metadata of the database should be written in JSON format and must follow the convention and rules for writing the metadata
- b. The C++ code should also follow the coding conventions set by the company and must include comment lines for all the functions written within the code
- c. The code is written in a path as set by the Visual Studio workspace and all the related files are written in the same project

Development Methods

The final output of the project depends upon the development methods that have been used. For best outputs, C++ is used for programming as our project is to build ODBC driver for which C++ is highly preferable. Python is also used for writing scripts to populate data. JSON module in python has to be installed to parse JSON and generate payload accordingly.

Architectural Strategies

This segment depicts the outline choices and techniques that influence the general association of the framework and its more elevated amount structures. These techniques will give understanding into the key deliberations and instruments utilized as a part of the framework engineering. This area tosses light on different outline procedures and choices which chooses the working of this venture and furthermore its larger amount structures. This area will give you the explanations behind the outline choices and systems that are taken to fabricate this venture.

Programming Language

The project is done using C++ because C++ is widely used in the development of data connectivity drivers and is easy to understand. Metadata of the database is written in JSON format. Data populating is done using Python code having JSON modules installed. Some tables are populated through UI for which Selenium has to be installed and Python scripts are written.

System Architecture

The JIRA ODBC Driver is built using the Simba Engine and writing the metadata of the database. Most important techniques used are enabling the authentication, parsing the metadata, fetching and adding the data using APIs. The architecture of the database is

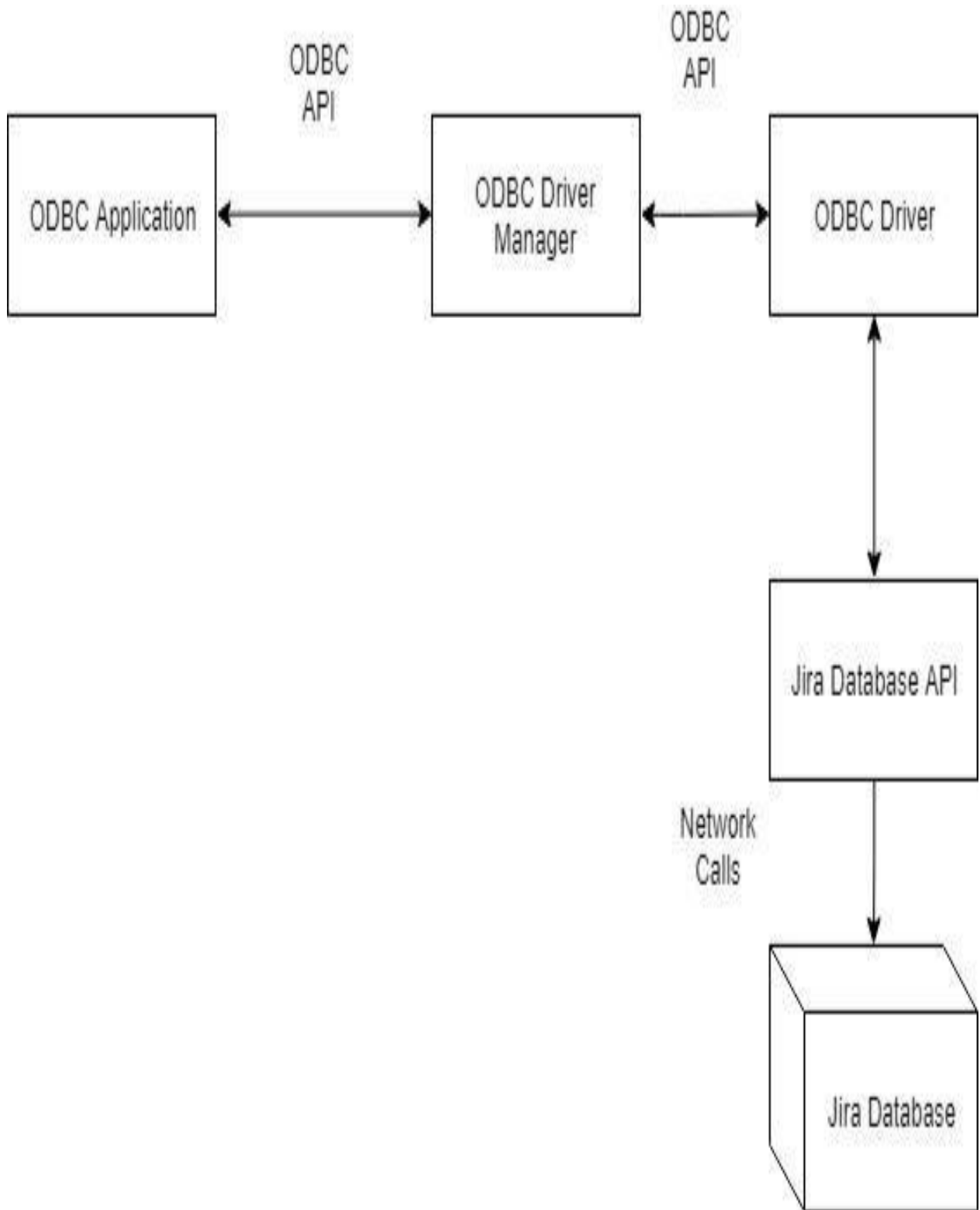


Figure 3.1: shows the system architecture of JIRA ODBC Driver

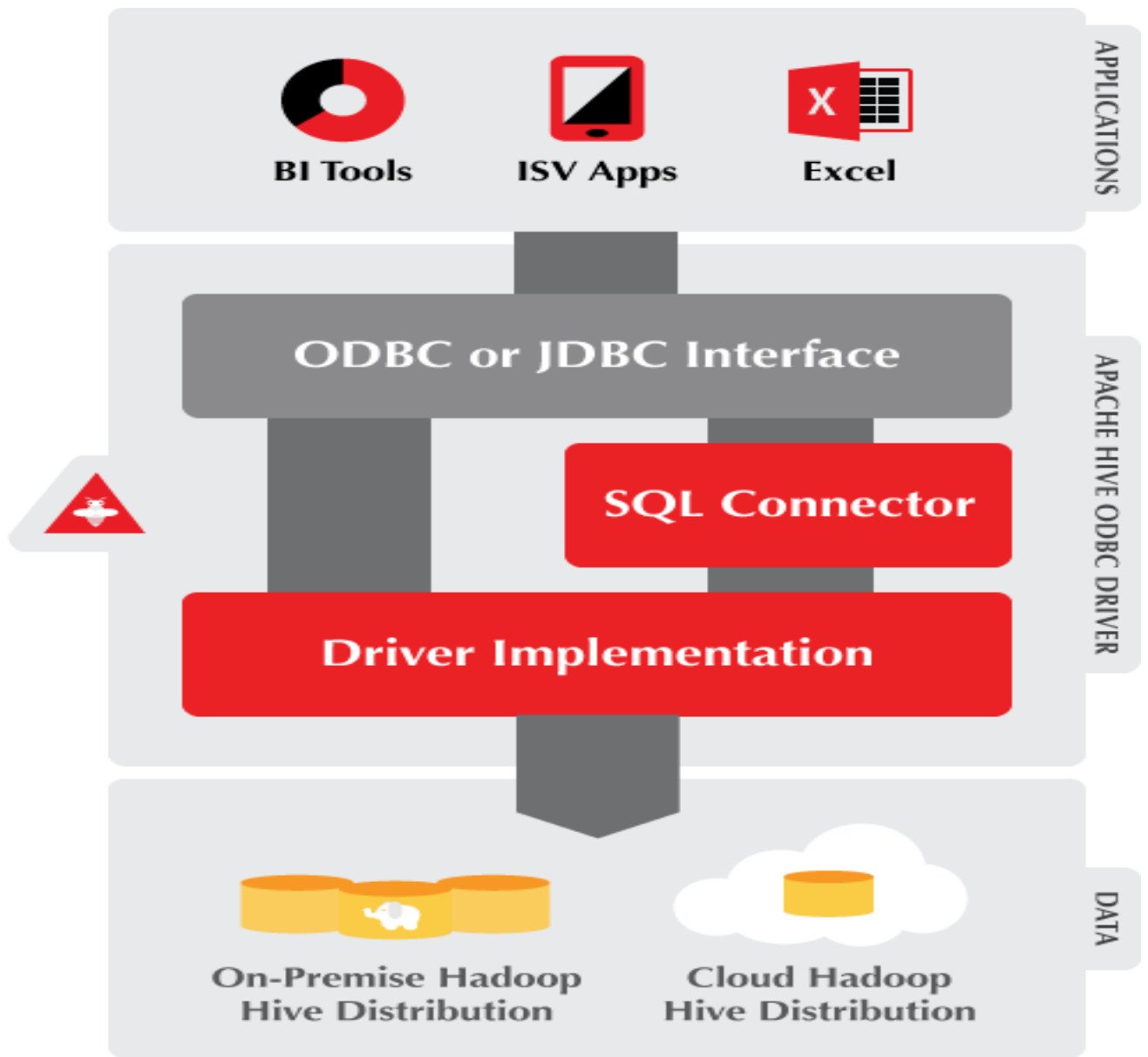


Figure 3.2: shows the High level design of the driver

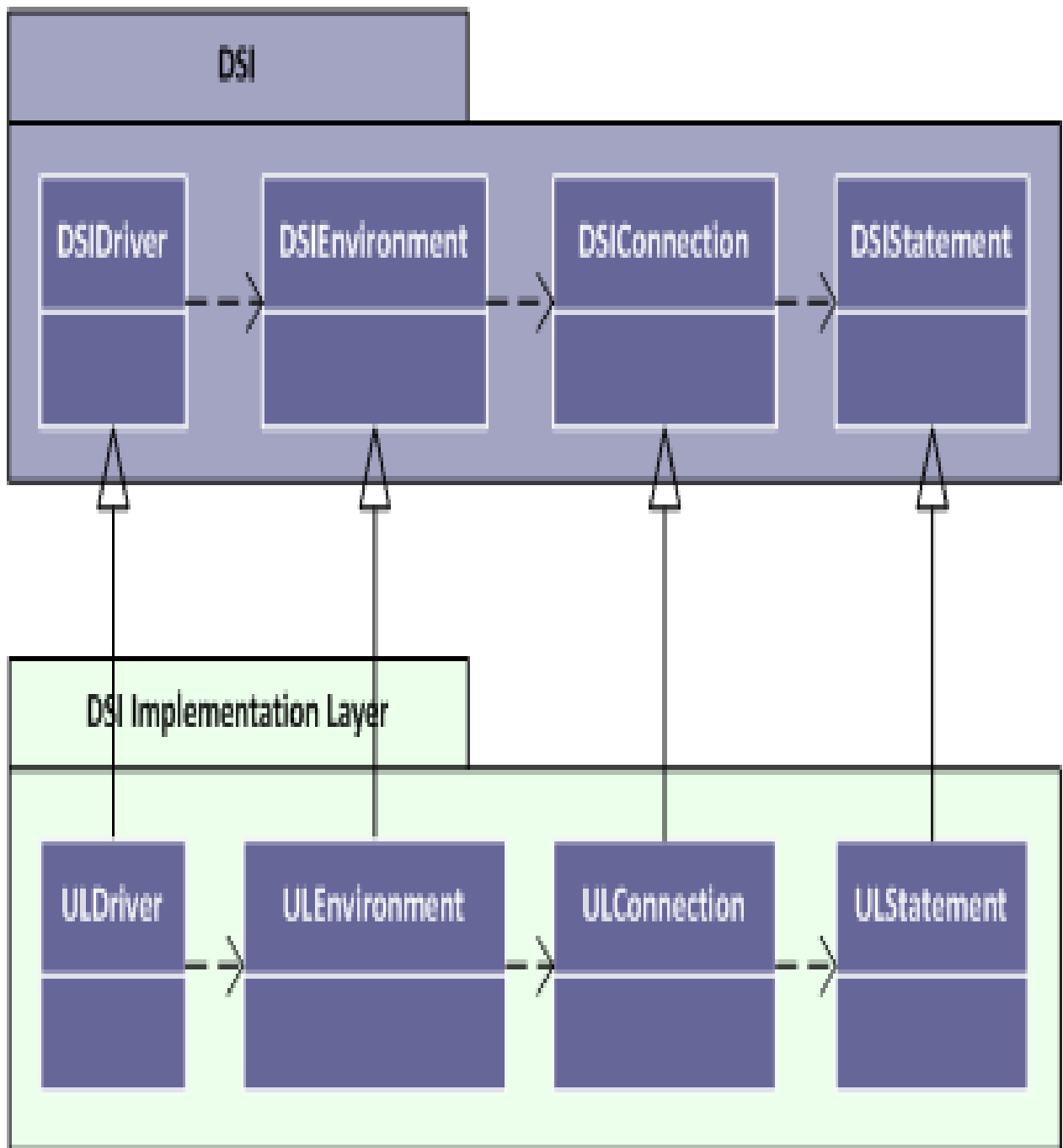


Figure 3.3: shows the Low level design of the driver

Data Flow Diagrams

A data flow diagram (DFD) is a pictorial speaks to the stream of information through a data framework. Rather than a flowchart that portrays the control stream of the program, a DFD demonstrates the stream of information through the course of the program. A DFD makes the way toward envisioning the organized plan, less bulky. Designed by Larry Constantine, the DFDs depend on Martin and Estrin's "information stream diagram" model of calculation.

It is normal practice to draw a setting level Data stream outline to start with, which portrays the collaboration between the outside elements and the framework. The DFD is intended to demonstrate the distinctive littler segments of the framework and is additionally used to highlight the stream of information between those parts. This setting level DFD is then exploded to show more detail of the framework being displayed.

The DFDs demonstrate the contribution to and yield from a framework. This incorporates data identified with what the information is and its stockpiling area. It doesn't demonstrate whether the procedures will be done consecutively or in parallel, nor any data about the planning of procedures. It additionally does not manage control stream of information.

Data Flow Diagram - Level 0

The level-0 of a block diagram is the fundamental level which speaks to the working of the entire venture in an extremely straightforward manner. The outer specialists introduce in level 0 chart are just the source, which is the beginning stage if the venture and the sink where it closes.

The entire framework is spoken to as only a solitary element. Figure 4.2 shows the level 0 Data flow diagram. Source and destination are the two external entities present here. This shows the working of the JIRA Driver.

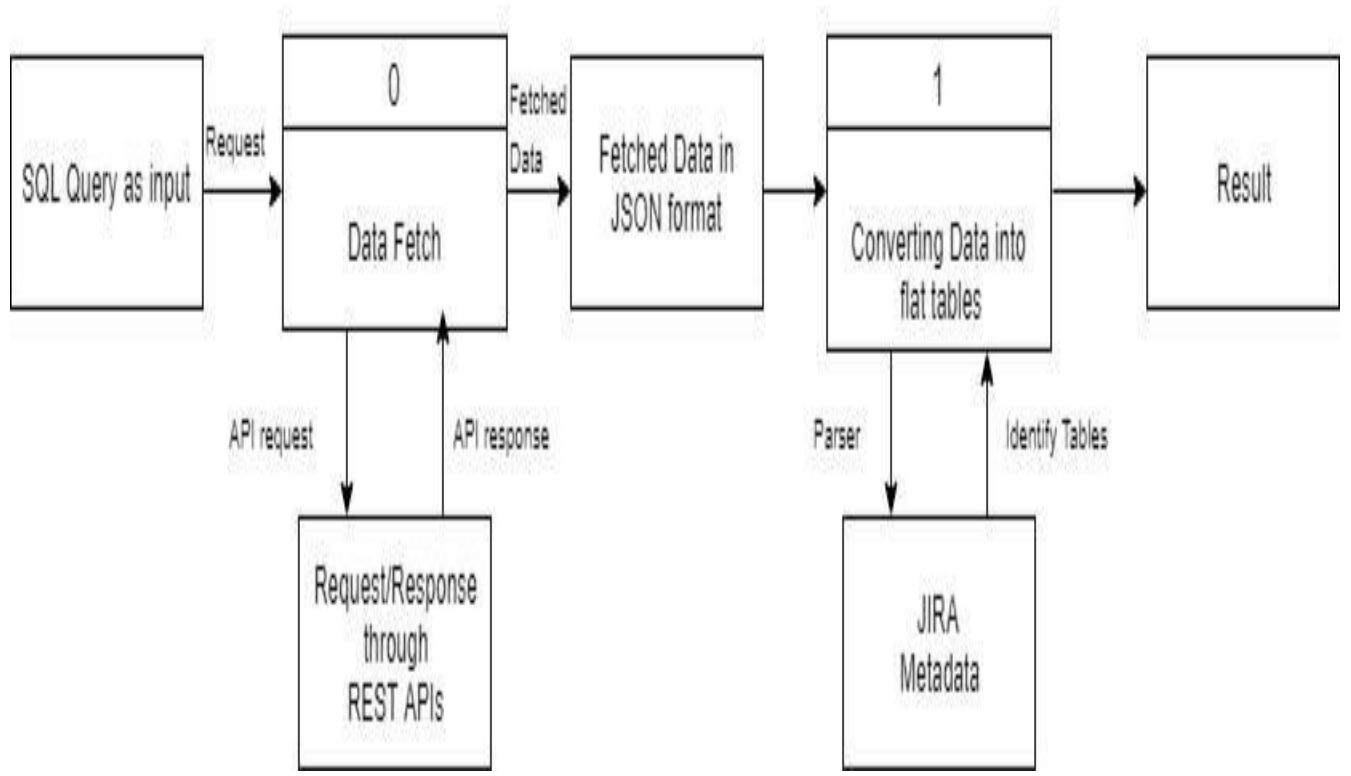


Figure 3.4 Level-0 Data Flow Diagram of the JIRA ODBC Driver

Data Flow Diagram - Level 1

The level 1 DFD is a context level DFD where the system is divided into constituent parts and the relationship between those parts.

The figure 3.3 shows the DFD - Level 1 for Data parser system from the DFD - Level 0. It shows how the system is divided into separate sub-systems.

Implementation of JIRA ODBC Driver

Implementation is the methodology of putting a result or technique into impact, or execution of an arrangement, thought, portrayal, display, outline, standard or calculation. In software engineering, usage is a comprehension of a specialized depiction, programming segment, or other PC framework. Numerous applications may exist for a given portrayal or quality.

Implementation Requirements

The software requires C++. The libraries required are Jackson to parse JSON, HttpClient, URIBuilder to build URI and send requests. Simba Engine SDK for development of the driver. Google Guava to use as an extension for C++ collections. Python 3 is required to write scripts which populate data to the database.

Details of Languages Used

The programming utilized as a part of this framework is C++. C++ is widely utilized abnormal state, broadly useful and dynamic programming dialect. Its plan rationality expresses comprehensibility of the code. One of the upsides of this programming dialect is that its punctuation gives its client to express ideas in less lines of code (LOC) than would be conceivable in dialects viz., C++. Develops outfitted by this dialect are intended to encourage clear projects on both a little and substantial scale C++ supports various programming standards which are comprehensive of practical programming, protest arranged and basic strategies. It includes a dynamic sort framework and programmed memory administration. C++ translators can be introduced on different working frameworks (OS) which will permit the usage of C++ code on an inconceivable accumulation of frameworks. For few of the most reasonable working frameworks, C++ code can be wrapped into discrete executable projects which will permit the appropriation of C++-construct programming for operation in light of those situations where the establishment of C++ mediator is a bit much. At the point when C++ source code is gathered then it is changed over to a middle of the road dialect byte-code which is then translated by an interpreter.

Few points of interest of utilizing Java in this framework are:

- C++ gives various in-manufactured capacities for a large number of the average functionalities required for Simba Engine SDK
- C++ yields huge numbers of the information structures which are appropriate for characteristic dialect handling.
- Both C++ and byte-code are thought to be stage free.

Platform Selection

A platform represents a software framework that permits the software to run. Generally programming languages, operating system, computer's architecture and relevant runtime libraries or graphical user interface are incorporated in any platform. Platform used for the system can be Windows, Linux or Mac OS X. The source code is "compiled" to a common language byte code which is then deciphered by a compiler, the C++, which then Interfaces that program with the C++ software libraries.

Platform Components

The platform used in the project comprises of machine JSON, Jackson and Selenium. All the libraries used in the project are open-source. Each of these are explained in detail in the following sections.

JSON

JavaScript Object Notation (JSON) is a lightweight data-interchange format. It is very easy for humans to read and write and also for machines to parse and generate. JSON is similar to text format that is language independent but uses conventions that are familiar to programmers of C-family of languages.

Jackson

The Jackson JsonParser class is a low level JSON parser. It is similar to the Java StAX parser for XML, except the JsonParser parses JSON and not XML.

The Jackson JsonParser works at a lower level than the Jackson Object Mapper. This makes the JsonParser faster than the Object Mapper, but also more cumbersome to work with.

Python Selenium

Python language bindings for Selenium WebDriver. The selenium package is used to automate web browser interaction from Python. Several browsers/drivers are supported (Firefox, Chrome, Internet Explorer, PhantomJS), as well as the Remote protocol.

Coding Conventions

Coding conventions are an arrangement of dialect particular rules that suggest well known great practices, programming style and strategies for code written in that language. Comments, variable and strategy naming tradition, record association and basic programming standards are secured under coding conventions. Adhering to rules enhances the coherence of the code. The code can be effortlessly comprehended by different developers utilizing a similar dialect, making programming updates and upkeep by others less demanding. Coding traditions are only rules and are authorized neither by compilers nor translators.

Line width: We will use a maximum line width of 100.

- Note: If you are using Eclipse, you can set a visual line under Window > Preferences > General > Editors > Text Editors. Check the "Show print margin" and set the "Print margin column" to 100.

File Ordering: The order of the components in the file will follow the generic C++ standards:

- copyright header,
- package statement
- import statements
- class comments
- class/interface declarations (i.e. variables, methods, etc)

Copyright: We will use the standard copyright headers on the top of ALL of our source code files.

Variable/Method Ordering: The ordering of variables/methods will follow the Java Standards. The general order is:

- static variables, ○ instance variables ○ constructors
- methods

Within each of those sections they are then ordered by visibility, from public to private.

Nested or Inner Classes: These are normally declared before any static or instance variables within a given class. Use these sparingly.

Indentation: Use spaces instead of the tab characters.

Note: If you are using Eclipse, you can set this preference under Window > Preferences > General > Editors > Text Editors. Change the "Displayed tab width" to 4 and check "Insert spaces for tabs".

Comments

There are two different types of commenting which are commonly used when programming in python. They are given below:

- a. **Single-line** comments start with # and continue until the end of line.
- b. **Multi-line** comments start with “ “ “ and end with ” ” ”.

Comments are utilized to elucidate the code and are not deciphered by the mediator. All through the code base remarks are utilized wherever required and they elucidate the code.

Summary

The chapter describes the implementation details of the JIRA ODBC Driver. The programming language and platforms selected along with the coding conventions adopted are mentioned.

CHAPTER-4

PERFORMANCE ANALYSIS / TESTING

Software Testing is a procedure in which the created programming is tried by the necessities given by the partners and customers. There are two sorts of testing black box testing and white box testing. In black box testing the product is tried by giving substantial and invalid data sources and checking the outcomes likewise, the information sources are as per the given prerequisites. In white box testing, the code of the product is tried to check for the inside usefulness of the same and it can likewise be corrected if broken. Here we are managing the black box testing and checking how the product capacities for the given prerequisites. Testing not just checks if the product capacities the way it ought to yet in the event that the product begins working the way it shouldn't, both are dealt with.

Testing Environment

Testing is done with respect to the requests.

6.1.1 Hardware Specifications

Below are the hardware specifications for the JIRA Driver:

- a. A PC with 16gb or more RAM

Software Specifications

Below are the software specifications required for the JIRA Driver:

- a. Operating system: Linux, Windows or Mac OSX
- b. Language: Python for populating data to the database, JSON for writing config.
- c. POSTMAN to hit the servers and test the Response Times

Unit Testing of Modules

Unit testing is a procedure used to validate that individual units of source code are working properly. A unit is the smallest testable part of an application. Any bugs encountered while executing the software can be localized to a particular feature of the tool by using unit testing.

Unit testing of JSON parser module

The JSON Parser module is tested, and the test case is tabulated below. The Parser is tested by giving the metadata of the database as input. The results are analysed by checking the catalogs from the test tool. It lists all the schemas, tables, primary keys and foreign keys in the database after parsing the metadata. The test was found to be successful. Table 4.1: references the test.

Result

Unit Test Case 1 is used to check if the JSON Parser module functions correctly. If the module gives the exact number of schemas, tables, primary keys and foreign keys as in metadata of the database, then the test is successful

Table 4.1: Unit Test Case 1: TPC trace Module

SI No. of test case :	ID1
Name of test :	JSON Parser Module Unit test
Item / Feature being tested :	JSON Parser Module

Description	JSON Parser module should be able to parse the metadata and list all the catalogs of the JIRA database.
Sample Input :	Metadata file of the JIRA database
Expected output :	List all the schemas, tables, primary keys and foreign keys of the JIRA database
Actual output :	All the schemas, tables and keys are fetched and listed.
Remarks :	The module is working properly

Unit testing of JSON Converter module

The JSON Converter module is tested, and the test case is tabulated below. This module is tested to check if the JSON response is parsed and converted into flat tables. The values fetched from the REST API responses is sent to the converter which converts the JSON and shows the result in flat tables format using catalogs from metadata. The test was found to be successful. Table 4.2: references the test.

Result

Unit Test Case 2 is used to check if the JSON Converter module functions correctly. If the module is able to convert the data in JSON format to flat tabled format. The test is success if the module is able to display the output for a query as data under some tables with rows and columns displayed.

Table 4.2: Unit Test Case 2: JSON Converter Module

SI No. of test case :	ID2
Name of test :	JSON Converter Module Unit test
Item / Feature being tested :	JSON Converter Module

Description	JSON Converter Module will convert the response which is in JSON format to flat tables.
Sample Input :	Response from REST API calls
Expected output :	The response from JSON which is schema less will be converted as data with schema.
Actual output :	The result for a query will be presented as data under some table with rows and columns
Remarks :	The module is working properly.

Unit testing of Simba Core module

The Simba Core module is tested, and the test case is tabulated below. This module is tested to check the working of the main algorithm and required functionality. The output of Simba Core module is the it parses the query to understand give result accordingly making necessary API calls. The test was found to be successful. Table 4.3: references the test.

Result

Unit Test Case 3 is used to check if the Simba Core module functions correctly. For a given SQL query the algorithm to understand the query, the table name and any additional attribute has to be work properly.

Table 4.3: Unit Test Case 3: Simba Core Module

SI No. of test case :	ID3
Name of test:	Simba Core Module Unit test
Item / Feature being tested:	Simba Core Module
Description	Simba Core Module parses the query to understand give result accordingly making necessary API calls
Sample Input:	Input is an SQL Query and metadata of the JIRA database which is the config for JIRA.
Expected output:	All the tests must pass as the JIRA Driver is running as accepted.
Actual output:	All the tests have passed.
Remarks:	The module is working properly.

Integration Testing of Modules

Integration testing is a logical extension of unit testing. Incorporation testing is a sensible expansion of unit testing. In its least difficult frame, two units that have as of now been tried are consolidated into a part and the interface between them is tried. Here, we will test every one of the modules portrayed before all the while. Coordination testing distinguishes issues that happen when units are joined. By utilizing a test plan that requires the testing of every unit and guaranteeing its suitability before joining them, we come to realize that any mistakes found when consolidating units are likely identified with the interface between units. This technique diminishes the quantity of conceivable outcomes to a far less difficult level of examination

Integration Testing of JSON Parser and Simba Core Module

Different parts of JSON Parser and Simba Core module are tested together, and the test case is tabulated below. The complete module is tested to check if it can perform the required pre-processing i.e. the grammar written for the JSON Parser module does not overlap with the Simba Core module. Both the modules have a different style of logging and the values to be looked for are different. So, upon integrating using the DLLs of Core with that of driver the results are analysed. In this test the models are tested extensively so that they yield same out even if they are imported to the same module. The test was found to be successful. Table 4.4: references the test.

Use Case

Integration test case 1 is used to check if the JSON Parser and Simba Core module works properly.

Table 4.4: Integration Test Case 1: Augmentation-Detection Module

SI No. of test case :	ID4
Name of test :	JSON Parser-Simba Core test
Item / Feature being tested :	JSON Parser and Simba Core.
Description	Both the parser and core are imported to the same module and the values are passed.
Sample Input :	SQL Query to insert, select or delete data from JIRA tables.
Expected output :	The algorithm should work fine to fetch results for the SQL query.
Actual output :	The results are fetched and sent to JSON converter .
Remarks :	The module is working properly.

Integration Testing of Simba Core and JSON Converter module

Different parts of Simba Core and JSON Converter module are tested together, and the test case is tabulated below. The complete module is tested to check if the results obtained from Simba Core for the query are properly converted into flat tables. The test was found to be successful. Table 4.5: references the test.

Use Case

Integration test case 2 is used to check if the Simba Core and JSON Converter module works properly.

Table 4.5: Integration Test Case 2: Simba Core and JSON Converter Module

SI No. of test case :	ID5
Name of test :	Simba Core and JSON Converter Module test
Item / Feature being tested :	Simba Core and JSON Converter Module
Description	Output from Simba Core is sent to JSON Parser which converts data into flat tables
Sample Input:	SQL Query and metadata.
Expected output:	Fetches data as flat tables.
Actual output:	Data with table name and its rows and columns
Remarks:	The module is working properly.

System Testing

System testing is testing that is conducted on complete, integrated systems to evaluate its compliance with the specified requirements. System testing falls within the scope of black box testing. As such, System testing must require no knowledge of the inner working of the code or design. The entire system and all its components are tested for their availability to work together to obtain a perfectly functioning system. The system test was found to be a success, and the Test case is tabulated below in Table 4.6.

Result

The System Test case 1 is used to ensure the compliance of all modules of the system. It tests to see if the various modules work in synchronization with each other to provide the expected results. The test is a success, if all the functions and components of the system work as advertised

Table 4.6: System Test Case 1

SI No. of test case :	ID6
Name of test :	JIRA Driver System Test
Item / Feature being tested :	Entire System functionality.
Description	Entire system is tested along with its modules- Simba Core, JIRA Parser and JIRA Converter.
Sample Input :	SQL Query or catalog functions
Expected output:	Results in the form of flat tables with rows and columns.
Actual output:	Driver fetches data and gave correct results
Remarks:	The module is working properly.

Results and Discussion

This discussion gives details about the project after testing different modules of the Caching Application Service. The results after testing shows if the requirements of the project are met and if the requirements are not satisfied, the reasons for the same are stated below. Towards the end, a brief discussion on future development is done.

Authentication

These are the results captured for the authentication.

- Authentication is working when there is an Internet connection.
- Appropriate errors are thrown if an error occurs

Parsing Queries

These results are captured after running various queries

- Incoming Queries are parsed and validated.
- Insert query is working for all the tables which support insert operation.
- Select query is working for all the tables which are readable.
- Update works for tables with update support.

Sending REST API Requests.

These results are captures when testing API calls

- Driver could convert SQL queries to Rest calls.
- Driver could make Rest calls successfully.
- Driver could transform insert SQL query to JSON body in the post request.

Processing the Result.

These results are captured when testing the output of the driver.

- It was able to read the json/xml/csv result.
- It was able to convert it to rows.
- It was able to handle nested response.

```
C:\Users\manjunath\AppData\Local\Temp\p4\MBodagal_perforce_1660\Drivers\Memphis\DataSource\Ira\Common\Maintenance\1.5\MDEF\driver-df4.mdef - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
BODagEngine.exe a2java * Anthony * addpy * logmap * add2py * driver-df4.mdef * generic-custommap * ira-rest-pugin.wad * projectpy * testpy * addpy * addpy * filterpy * iralpy * gcpy * grouppy *
9189 }
9190 }
9191 "Name": "CreateUser",
9192 "ProcedureType": "SIMPLE",
9193 "URL": "http://HOST_PORT/rest/api/2/user",
9194 "Headers": {
9195   "Content-type": "application/json"
9196 }
9197 "Method": "POST",
9198 "RequestBodyTransformRule": "BuildPayloadforCreateuser",
9199 "ProcedureParameters": [
9200   {
9201     "Index": 0,
9202     "Key": "User_Name",
9203     "SQLType": "SQL_VARCHAR",
9204     "Length": 1024
9205   },
9206   {
9207     "Index": 1,
9208     "Key": "Password",
9209     "DefaultValue": "",
9210     "HasDefaultValue": true,
9211     "SQLType": "SQL_VARCHAR",
9212     "Length": 1024
9213   },
9214   {
9215     "Index": 2,
9216     "Key": "Email Address",
9217     "DefaultValue": "",
9218     "HasDefaultValue": true,
9219     "SQLType": "SQL_VARCHAR",
9220     "Length": 1024
9221   },
9222   {
9223     "Index": 3,
9224     "Key": "Display Name",
9225     "DefaultValue": "",
9226     "HasDefaultValue": true,
9227     "SQLType": "SQL_VARCHAR",
9228     "Length": 1024
9229   },
9230   {
9231     "Index": 4,
9232     "Key": "Application Keys",
9233     "DefaultValue": "",
9234     "HasDefaultValue": true,
9235     "SQLType": "SQL_VARCHAR",
9236     "Length": 1024
9237   }
9238 ],
9239 "ResultType": "JSONTABLE",
9240 "ResultTable": {
9241   "Columns": [
9242     {
9243       "Name": "Active",
9244       "Metadata": {
9245         "SQLType": "SQL_BIT"
9246       },
9247       "SvcRespAttr_listResult": "active"
9248     },
9249     {
9250       "Name": "Application_Roles_Max_Results",

```

Image 4.1: Showing the Metadata of the tables

Column Name	SQL Type	Filter Name	Updatable	Nullable	Column Type	Foldable	Foldable Notes
Issue_Type	SQL_INTEGER(signed)	avatarId	FALSE	TRUE		FALSE	
Avatar_Id	SQL_INTEGER(signed)	description	FALSE	TRUE		FALSE	
Description	SQL_INTEGER(signed)	iconUrl	FALSE	TRUE		FALSE	
Icon_Url	SQL_INTEGER(signed)	id	FALSE	TRUE	PK(pk_IssueType)	FALSE	
Name	SQL_INTEGER(signed)	name	FALSE	TRUE		FALSE	
Self	SQL_INTEGER(signed)	self	FALSE	TRUE		FALSE	
Subtask	SQL_INTEGER(signed)	subtask	FALSE	TRUE		FALSE	
Value	SQL_INTEGER(signed)	value	FALSE	TRUE		FALSE	
Key	SQL_INTEGER(signed)	key	FALSE	TRUE		FALSE	
Self	SQL_INTEGER(signed)	keys.self	FALSE	TRUE		FALSE	
Start_At	SQL_INTEGER(signed)	startAt	FALSE	TRUE		TRUE	
Total	SQL_INTEGER(signed)	total	FALSE	TRUE		FALSE	
Max_Results	SQL_INTEGER(signed)	maxResults	FALSE	TRUE		TRUE	
comments_Index	SQL_INTEGER(true)	comments_Index	FALSE	FALSE	PK(pk_IssueTypeOrKeyCommentCo	FALSE	
comments_author.active	SQL_INTEGER(true)	author.active	FALSE	TRUE		FALSE	
comments_author.avatarUrls	SQL_INTEGER(true)	author.avatarUrls	FALSE	TRUE		FALSE	
comments_author.displayName	SQL_INTEGER(true)	author.displayName	FALSE	TRUE		FALSE	
comments_author.emailAddress	SQL_INTEGER(true)	author.emailAddress	FALSE	TRUE		FALSE	
comments_author.key	SQL_INTEGER(true)	author.key	FALSE	TRUE		FALSE	
comments_author.name	SQL_INTEGER(true)	author.name	FALSE	TRUE		FALSE	

Image 4.2: Showing the Schema of all the tables

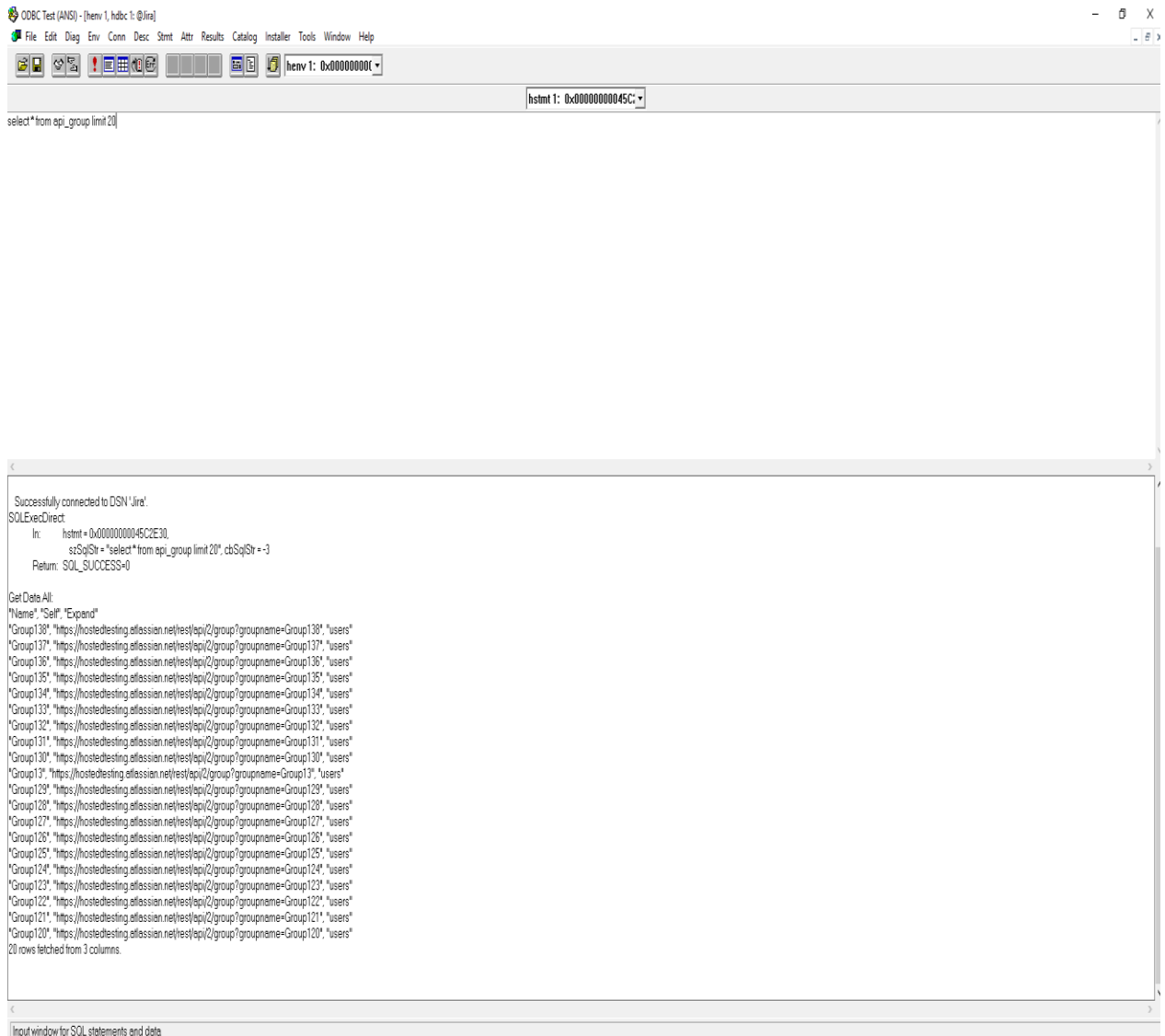


Image 4.2: Showing the results of the queries in ODBC Test Tool

Discussion and Inference

The project is a C++ implementation so it will not have a front end and it will be platform independent. We can import the driver in the client application and use it. The driver manager acts as a messenger between driver and the client application. The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

Summary

The chapter is associated with the various unit, integration and system tests conducted on the JIRA ODBC Driver System. The input parameters, output results and descriptions of each test is mentioned in the table representing the test. The system passed all the tests and helped in assuring the functionality and quality of the system

CHAPTER-5

CONCLUSION

Conclusion

The motivation of the driver was fulfilled by developing the software that takes SQL queries for large amount of data and connecting the driver to the BI tools like PowerBI and Netezza which enables the clients to perform analysis on their data and come up with business models which can help them to improve their businesses. Using Simba EngineX one can build their own driver for their database as the Simba Engine can be used for building driver which can be highly scalable. The basic building block of the JIRA driver are the APIs of the JIRA database, metadata and the Simba Engine and it was successfully understood in the course of the project

Limitations

Limitations of the project are it is required to understand and design the schema of the JIRA database and there is no automated system to get the schema of the database. The limitations in our project are found in the techniques used and can be taken as challenge for future enhancements:

Some important limitations as follows:

- The JIRA server is too slow as it takes some time to get the responses for the requests sent.
- Some tables are too complex to understand and have high dependencies which takes a lot of time to analyse.
- APIs are not present for all the tables of JIRA database and data loading has to be done through UI sometimes.
- The performance of the driver is also largely dependent on the host system.

Future Enhancements

- The JIRA database could be enhanced to send responses in shorter time period for the requests sent.
- A more detailed documentation about the tables, it's structure and the dependencies can be provided.
- Upon having the APIs for all the tables in the database there will be no need for inserting data through UI.

REFERENCES

- [1]. Evan McGlenn, "Blueprint Lets 1-2-3 Access Outside Data", InfoWorld, 4 April 2012.
- [2]. Harindranath, G, Joze Zupancic (2007). New perspectives on information systems development: theory, methods, and practice. Springer. p. 451. ISBN 978-0-306-47251-0. Retrieved 2010-07-28. The first ODBC drivers used the SIMBA query processor, which translated calls into the Microsoft Jet ISAM calls, and dispatched the calls to the appropriate ISAM driver to access the backend
- [3] Idehen, Kingsley (*October 1994*). "ODBC and progress V7.2d". Usenet *Newsgroup comp.databases*. Retrieved 13 December 2013.
- [4] Anderson, Andrew (2003-06-20). "Open Database Connectivity in Jaguar". O'Reilly MacDevCenter.com. O'Reilly Media, Inc. Retrieved 13 December 2013.
- [5] Sellers, Dennis (2001-07-17). "ODBC SDK update out for Mac OS Classic, Mac OS X". MacWorld. IDG Consumer & SMB. Retrieved 13 December 2013.
- [6] Wang Jikui, "The Road to be a storage expert - storage from entry to the master", *Tsinghua University Press*, 2016.
- [7] Zheng Guochang, "Synchronous Relication and Asynchronous Replication", *Modern Computer*, vol. 04, 2015.