End-Of-Study project dissertation

For achieving the title of

IT engineer

Subject

improving Oracle's Resource Manager on the PGX framework

Supported by:
TAUIL Youssef

Under the direction of:
Mr. BALOUKI Youssef
Mr. SOUKRAT Anas

College year : 2019 - 2020

Dedication

"IN THE NAME OF ALLAH, THE MOST GRACIOUS AND MERCIFUL, THE BENEFICENT"

First of all I would like to bend over my head in front of **ALMIGHTY ALLAH** . It was **HE** who direct me in every stage of my life, and who bestow me with capability to accomplish the best and helped me to conquer every attempt and difficulty in every field of life. Without His help, we cannot accomplish any objectives in our lives.

To my dear mother Mahdia, To my dear father Mohamed

No dedication can express my respect, love eternal and consideration for the sacrifices you have made for my education and well-being. I thank you for all the support and love you have given me since I was a child and I hope that your blessing will always be with me. May this modest work be the fulfillment of your many wishes, the fruit of your countless sacrifices, although I can never repay you enough. May God, the Most High, grant you health, happiness and long life, and see to it that I never disappoint you.

To my big sister Safaa

As a token of my fraternal affection, of my deep tenderness and gratitude, I wish you a life full of happiness and success.

To my brothers and sisters

No dedication is enough to all my GI14 brothers and sisters who took me gently by the hand to cross together the years of engineering, and for their company and unforgettable moments spent together.

acknowledgment

First of all, praise to **ALLAH** who has guided us on the right path throughout the work, and inspired us to take the right steps and reflexes, without his mercy, this work wouldn't have come to pass.

I would like to express my sincere thanks to our head of the sector, **Mr. REDOUANE Adil**, for all the great efforts he has made throughout our course to enable us to reach this point, and may he find in this modest work the expression of my gratitude.

I warmly thank **Mr. BALOUKI Youssef**, my supervising professor at the Mohammed VI International Academy of Civil Aviation, for having continuously listened to me; his availability, his benevolence, his involvement, and above all his valuable recommendations have been invaluable to me.

I would like to thank **Mr. EZ-ZARGHILI Mohamed**, the project manager, for having generously given us his time with little flexibility and for having guided us with a lot of patience.

A deep thanks is addressed to**Mr. WELD Alexander**, our Head of the PGX.SM team and my functional supervisor and **Mr. SOUKRAT Anas** our technical supervisor, for their collaboration and for the advice and information he has given me, shared with me with unparalleled sympathy, as well as for giving me his time to supervise me despite his occupations.

I would like to express my gratitude to **Mr. HILLOULIN Damien** and
Mr. TRIGONAKIS Vasileios, members of my team who made this training course enriching by their availability and good mood, and granted me all the necessary assistance for the success of this project.

Finally, I would like to thank everyone who contributed in any way to the realization of this work. May this work bring full satisfaction to all those who read it.

A thousand thanks.

Abstract

This document is a summary report of my graduation project within Oracle Morocco which is part of the contribution to customer resource management for the benefit of Oracle's PGX project.

It is more specifically a question of finding a solution to manage server resources, optimizing as much as possible the resources used to support the graphs on the server, to implement new functionalities, write the necessary tests, and complete the PGX library with new APIs.

For the realization of this project, we used JAVA as a development language and GRADLE to manage dependencies. In parallel, we used the ORACLE database with SQLPLUS and PGQL query languages.

The development of the different parts of the project was managed according to an approach that began with a functional and technical study, a detailed design of the project, as well as a realization of unit tests and which ended with a demonstration and presentation of the work carried out.

Keywords: GRADLE - JAVA - PGX - PGQL - SQLPLUS - API .

Résumé

Ce présent document s'agit d'un rapport synthétique de mon projet de fin d'études au sein d'Oracle Morocco qui s'inscrit dans le contexte de la contribution au management des ressources des clients au profit du projet PGX d'oracle.

Il s'agit plus spécifiquement de trouver une solution pour manager les ressources de serveur, d'optimiser au maximum les ressources utilisées pour supporter les graphes au serveur, implémenter des nouvelles fonctionnalités, écrire les tests nécessaires, et compléter la bibliothèque PGX par des nouveaux APIs.

Pour la réalisation de ce projet, nous avons fait recours au JAVA en tant que langage de développement et GRADLE pour assurer la gestion de dépendances. En parallèle, nous avons utilisé la base de données d'ORACLE avec les langages de requêtage SQLPLUS et PGQL.

Le développement des différentes parties du projet a été géré selon une démarche qui a commencé avec une étude fonctionnelle et technique, une conception détaillée du projet, ainsi qu'une réalisation des tests unitaire et qui s'est achevée avec une démonstration et présentation du travail réalisé.

Mots clés : GRADLE - JAVA - PGX - PGQL - SQLPLUS - API .

Glossary

Abréviations	Spécifications
ERP	Entreprise Resource Planning
CRM	Customer Relationship Management
HPC	Hight Performance Computing
API	Application Programming Interface
GC	Garbage Collector
DS	Data Structure
OOM	Out Of Memory
OOP	Object Oriented Programming
REST	Representational State Transfer
RDBMS	Relational Data Base Management System
SQLPLUS	Structured Query language PLUS
UML	Unified Modeling Language
SSL	Secure Sockets Layer
TLS	Transport Layer Security
LDAP	Lightweight Directory Access Protocol
PGX	Parallel Graph analytiX
PGQL	Parallel Graph Query Language
DB	Data base
ETL	Extract Transform Load

List of Tables

List of Figures

Contents

General introdution

Graphs are everywhere. Many of us use them or come across them daily. In its essence, a graph is an abstract data type that requires two basic building blocks: nodes and vertices. A graph utilises the basic idea of using vertices to establish relationships between pairs of nodes. In terms of applications.many real world relationships are best modeled using graph structures.

Graphs indeed offer a powerful data structure because it is extremely flexible, allowing not only to save data, but also and above all the relationships between them. Such data structuring requires an efficient and parallel implementation of the graph analysis tools. From this need arises Oracle's PGX project, a framework for fast, parallel and central memory analysis of graph processing algorithms, which constitutes the framework of my End of Studies Project. My internship missions, as we will see later in this report, consist in effect of making a series of optimizations and corrections to the PGX engine, and supporting new APIs for the engine.

Oracle recently addressed this issue in order to provide its main server, which represents the main product, with this functionality. His efforts in the matter resulted in a prototype on behalf of Resource Manager. The problem manifested in the memory and server performance part and it is in this context that my PFE fits.

This dissertation is organized into four main chapters and describes the objectives and the main lines of our end of studies project.

The first chapter is devoted to the presentation of the host organization Oracle Labs. He will also present the general context of the project and the problematic proposal accompany with the required objectives to resolve the problem, and also the management of the project.

The second chapter will deal with the technical context of the internship. Among other things, it will include an in-depth analysis of the prototype and the interact components in order to identify the limits in terms of its functionality.

The third chapter will describe the design and implementation of the proposed solution on all levels of the existing prototype. He will therefore draw up the technical architecture of the supported solution and the acceptance criteria of our prototype with the test environment used during the development stage.

The fourth and last chapter deals with the realisation and demonstration of the implementation that was presented in the previous chapter. He will present the several steps for the delivery of the project. This chapter also presents detailed demonstration created to illustrate our work and the nature of the changes made. At the same time, we will evaluate the other tasks to our prototype that I hold during the internship journey.

Finally, we will close this memory with an appropriate conclusion without forgetting to mention the horizon of the project and the enormous potential that holds this work.

Chapter 1

Context and goals of the project

The main goal of this chapter is to situate the context of the project. We will introduce the project frame, the problematic, the goals, and the taken steps to achieve it.

***

1.1 Project framework

In order to better understanding of the internship, we define carefully the project framework as well as the necessary issues and achieved goals.

1.1.1 Host organisation

My end-of-study internship took place within Oracle Corporation and more particularly on Oracle Labs entity mainly dedicated to research.

1.1.1.1 Oracle Corporation

Figure 1.1 : – Logo Oracle.

Co-founded in 1977 by Larry Ellison, Oracle Corporation is an American multinational based in Redwood Shores in California and specialized in computer technologies, it is mainly known for its relational database management system (RDBMS).

The company mainly specializes in development and marketing software and database technologies for which it's market leader,CLOUD systems and ERP integrated management software, customer relationship management and managing the SCM supply chain.

In 2015, Oracle was the second largest software maker in terms of revenues, after Microsoft. Here's a table describing the key figures that characterize Oracle statistics:

Table 1.1 - Key figures of Oracle corporation.

Effective	137 000 (2018)
Foundation	1977
Capital	186 billions USD (2019)
Revenue	US 39.83 billions USD (2018)

Oracle's diversity to other areas is becuase of its strategic acquisitions which we can list :

• **PeopleSoft et Siebel Systems :** Publisher of CRM software and tools in 2005 and 2006 respectively.
• **BEA Systems :** Development software platform (2008).
• **NetSuite :** ERP Saas platform (2016).
• **Aconex :** Specialize on Cloud Computing(2017).

Oracle's commitment to RD is a driving factor in the development of technologies that have kept Oracle at the forefront of the computer industry. Although many of Oracle's leading-edge technologies originate in its product development organizations, Oracle Labs is the sole organization at Oracle that is devoted exclusively to research.

1.1.1.2 Oracle Labs

The Mission of Oracle Labs is straightforward: Identify, explore, and transfer new technologies that have the potential to substantially improve Oracle's business.

Oracle Labs

Figure 1.1 : – Logo Oracle Labs.

Oracle Labs researchers look for novel approaches and methodologies, often taking on projects with high risk or uncertainty, or that are difficult to tackle within a product-development organization. Oracle Labs research is focused on real-world outcomes: our researchers aim to develop technologies that will someday play a significant role in the evolution of technology and society. For example, chip multi-threading and the Java programming language grew out of work done in Oracle Labs.

Oracle Labs maintains a balanced research portfolio with four major approaches:

1. **Exploratory research :**
   Bringing in the best and brightest in their fields to pursue their ideas within domains relevant to Oracle.
2. **Directed research :**
   Working in collaboration with product teams on difficult, future-looking problems outside the scope of the product release life-cycle, but driven by product requirements.
3. **Consulting :**
   Providing unique expertise that is useful in smaller engagements across many product organizations.
4. **Product incubation :**
   Providing a place to grow new products resulting from Oracle Labs research. Incubation is necessary when research results do not have a natural home (as is often the case with research across product areas), or where more risk needs to be eliminated from the work to demonstrate its value.

1.1.1.3 Oracle Labs projects

The labs is currently working on various projects, each focused in one area of different computer research, among which we can list:

- **Callisto :**

Modern h/w and s/w looks radically different to the HPC and server workloads which have traditionally motivated parallel programming: Individual machines are internally more complex with multiple processor types, NUMA domains and hot-swap components. Workload resource demands are also more complex, with bursty behavior over short timescales, and an ability to elastically use more/fewer resources as they are available. Furthermore, the need to consolidate workloads on large machines or clusters brings a renewed need to handle resource management on behalf of multiple jobs and multiple users.

- **Frappe :**

Frappe is a new tool that helps developers understand and navigate dependencies in their code. It aims to address the limitations of tools that developers normally use when they try to find a particular code in large code-bases. From a codebase, Frappe extracts a dependency graph that lets developers ask questions about the code, and then shows them the answers. Frappe uses the context of the question to give answers that are much more precise than would otherwise be possible.

- **Project Q :**

is oriented toward the graph analysis field, and all the necessary components for this purpose, among the sub-projects that we can list :

∗ **PGX :**

PGX is a toolkit for graph analysis that supports both running algorithms such as PageRank on graphs, and performing SQL-like pattern-matching on graphs, using the results of algorithmic analyses. Algorithms are parallelized for extreme performance. The PGX toolkit includes both a single-node in-memory engine, and a distributed engine for extremely large graphs. Graphs can be loaded from a variety of sources including flat files, SQL and NoSQL databases and Apache Spark and Hadoop; incremental updates are supported.

∗ **Walnut :**

The Walnut project leverages GraalVM in data processing engines, focusing on the rapid embedding of new programming languages (JavaScript, Python, Java, Ruby, R and others) and the use of runtime code generation and speculative optimization in query processing.

1.1.1.4 Expansion strategy

Oracle Labs is present by its individual researchers and experienced in several regions including America, Australia, Europe and the United Kingdoms.

This geographic distribution allows Oracle Labs to take advantage of a multitude of scientific and technical talents existing around the world, and enables researchers Labs to collaborate with colleagues from many sectors and universities.

Oracle Labs recently moved to Morocco in Casablanca, and launched its strategy expansion to expand its workforce to replicate success models previous labs, notably that of Mexico which has more than 1000 collaborators currently.

1.1.2 Project overview

1.1.2.1 PGX and Graph Analysis

The following figure depicts an overview of using PGX for graph analysis.

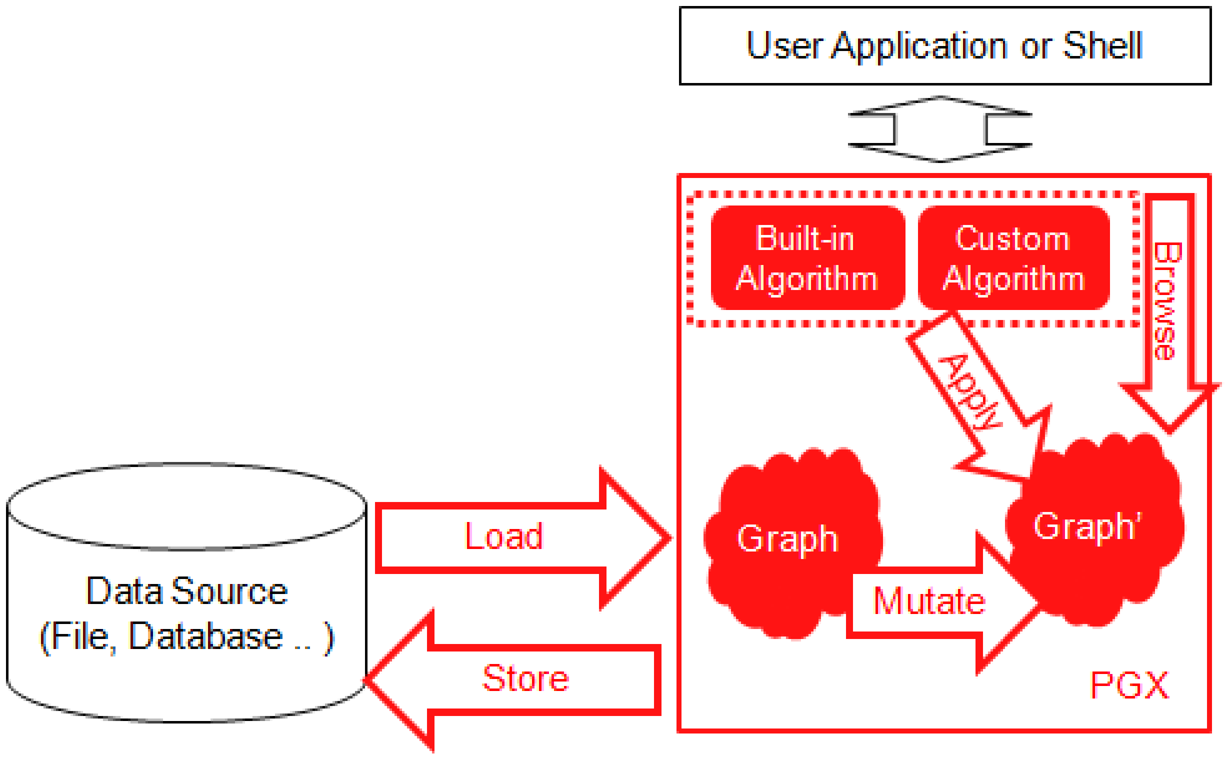


Figure 1.3 : – PGX overview.

PGX is a fast, parallel, in-memory graph analytic framework. PGX allows the user to do the following things in an easy and efficient manner:

- **Loading graphs into memory**:
  - ∗ PGX is an in-memory graph analytic framework that needs to load a graph instance into main-memory before running analytic algorithms on the graph. PGX supports a few popular graph file formats for convenient data loading.

- **Running built-in graph algorithms**:
    - ∗ PGX provides built-in implementations of many popular graph algorithms. The user can easily apply these algorithms on their graph data sets by simply invoking the appropriate methods.
- **Running custom graph algorithms** :
    - ∗ PGX is also able to execute custom (i.e. user-provided) graph algorithms. Users can write up their own graph algorithms with the Green-Marl DSL and feed it to PGX. The provided Green-Marl program is transformed for PGX using a parallelizing compiler.
- **Running graph pattern matching queries** :
    - ∗ Along with running graph algorithms, finding sub-graphs of interest to users' is a crucial task in graph analysis. PGX provides a graph pattern matching feature in which a query is written in an SQL-like declarative language, PGQL. PGQL queries are processed in a highly efficient manner on top of PGX.
- **Mutating Graphs** :
    - ∗ Complicated graph analysis often consists of multiple steps, where some of the steps require graph mutating operations. For example, one may want to create an undirected version of the graph, renumber the vertices in the graph, or remove repeated edges between vertices. PGX provides fast, parallel built-in implementation of such operations.
- **Browsing and exporting results** :
    - ∗ Once the analysis is finished, the users can browse the results of their analysis and export them into the file system.

1.1.2.2 PGX teams-work

One of the necessary strategy on each running project to pursue, is to separate the workflow on multiple teams, each team work on a specific field.

On our project PGX, we got so many fields to lead and advance them with new technologies, like algorithmic, security, and Machine Learning.

On my internship, I got interact with two main teams :

- **PGX.SM :** The Single Machine part of PGX, is a graph processing engine which run on single-node in memory engine.

- **PGX.D :** The Distributed part of PGX, is a graph processing engine which is distributed over the memory of multiple machines in a cluster.

PGX.SM, the team that I joined during my internship, focuses on improving the PGX engine through supporting new features and discussing variant solutions for the problems .

PGX.D, the team we collaborate with on the supported improvements and solutions to make a consistent architecture for both engines on the PGX Framework.

1.1.3 Problematic

On the road of building such a mature project as PGX, the legacy members passed through many obstacles to achieve that level.

Recently, the PGX framework support those services on new era of graphs, which is the heterogeneous graphs (partitioned graphs model), and since that new model hold huge amount of memory, the PGX.SM team come up with the idea of supporting a Resource Manager on the shared memory engine part, that problem can be a bottleneck on every project because the resources always limited and using them keep growing with the project progress, but on our part, it's inevitable problem to resolve and require a lot of work.

As member of PGX.SM team, my mission along the internship is to hold a new Resource Manager prototype on the shared memory engine, and manage the server memory, that problem it's relative to the PGX.SM team, which is not an issue for the distributed team since they had multiple machines option.

1.1.4 Objectives

On my project, the main goal is to build a first version for the Resource Manager prototype, that will achieve the given requirements, as a global improvement we need to increase server performance, and maintain the high-availability of the server, then control the users behaviour and manage the memory resources by newly introduced features.

To achieve those requirements, we set up the given goals as the vision of the project, at first we need to specify a strategy to limit the resources by supporting a tracker on the memory allocations, then we provide a process to trigger the graph GC on the engine.

1.2 Project management

1.2.1 Internship Procedure

My internship with Oracle was carried out using the Kanban agile method.
At first, we go through several training stages before having access to the source code and the rest of the platforms, then we attend several meetings for better integration with the team, and finally we got free access to two of the best e-learning platforms ; Oreilly and LinkedIn Learning .

1.2.2 Kanban methodology

A management methodology is necessary to carry out the project. This provides an effective monitoring on several stages of the project, from planning to implementation, while optimizing the efficiency and profitability of team members.

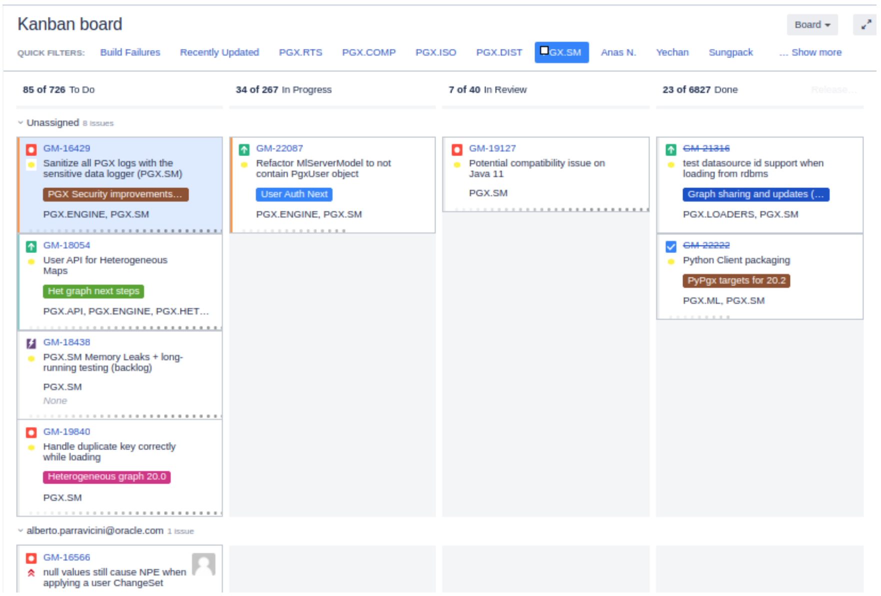
The PGX.SM team, as well as all Oracle Labs teams adopt the Kanban methodology. Derived from the Japanese term "sign card", Kanban is a an agile approach widely used in the IT world. This method avoids overloading teams, seeks to stimulate collaboration for continuous improvement of the system.

The Kanban approach offers the use of a task table to increase visibility of the team's work. Tasks are positioned in corresponding columns at the workflow stages. These flows represent the life cycle of a task before being validated by the team and then by management to finally being integrated.

Jira

Jira is the tool used to assign and track tickets according to the Kanban methodology, and allows the visualization of the methodology. Each ticket can have several sub-tickets that presents details and description of the task at hand.

the figure below represented the Kanban board for our PGX.SM team :



Kanban board

Board

QUICK FILTERS: Build Failures | Recently Updated | PGX.RTS | PGX.COMP | PGX.ISO | PGX.DIST | GX.SM | Anas N. | Yechan | Sungpack | ... Show more

85 of 726 To Do	34 of 267 In Progress	7 of 40 In Review	23 of 6827 Done

Unassigned 8 issues

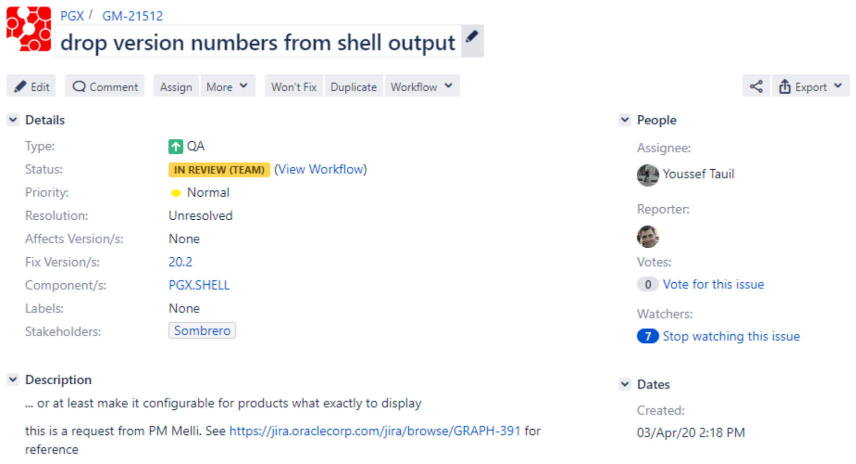
To Do	In Progress	In Review	Done
GM-16429 Sanitize all PGX logs with the sensitive data logger (PGX.SM) · PGX Security improvements... · PGX.ENGINE, PGX.SM	GM-22087 Refactor MlServerModel to not contain PgxUser object · User Auth Next · PGX.ENGINE, PGX.SM	GM-19127 Potential compatibility issue on Java 11 · PGX.SM	~~GM-21316~~ test datasource id support when loading from rdbms · Graph sharing and updates (... · PGX.LOADERS, PGX.SM
GM-18054 User API for Heterogeneous Maps · Het graph next steps · PGX.API, PGX.ENGINE, PGX.HET...			~~GM-22222~~ Python Client packaging · PyPgx targets for 20.2 · PGX.ML, PGX.SM
GM-18438 PGX.SM Memory Leaks + long-running testing (backlog) · PGX.SM · None			
GM-19840 Handle duplicate key correctly while loading · Heterogeneous graph 20.0 · PGX.SM			

alberto.parravicini@oracle.com 1 issue

GM-16566 null values still cause NPE when applying a user ChangeSet

Figure 1.4 : – Kanban Board.

The Kanban dashboard for the PGX.SM team is made up of several tickets.Each label can be viewed for more details on the assigned task. The label below relate to manage our shell output. The In Review status represents the last step before the final validation of the application.



PGX / GM-21512

drop version numbers from shell output

Edit | Comment | Assign | More | Won't Fix | Duplicate | Workflow | Export

Details

Type:	QA
Status:	IN REVIEW (TEAM) (View Workflow)
Priority:	Normal
Resolution:	Unresolved
Affects Version/s:	None
Fix Version/s:	20.2
Component/s:	PGX.SHELL
Labels:	None
Stakeholders:	Sombrero

Description

... or at least make it configurable for products what exactly to display

this is a request from PM Melli. See https://jira.oraclecorp.com/jira/browse/GRAPH-391 for reference

People

Assignee: Youssef Tauil

Reporter:

Votes: 0 Vote for this issue

Watchers: 7 Stop watching this issue

Dates

Created: 03/Apr/20 2:18 PM

Figure 1.5 : – Jira ticket.

Confluence

Confluence is a collaboration tool used to help teams collaborate and share knowledge efficiently. In Confluence, content is created and organized using spaces, pages, and blogs. As a new member, it's so helpful to keep track of the project, and each team member need to write their achievements at the level of platform.
Members should also summarize the tasks done during a work as weekly reports, each weekly page contains the following details :

- **Accomplishments :** the done tasks during the week.
- **In Progress :** tasks still on process of executing.
- **To Do :** future tasks need to be started next week.
- **Blocking :** List of unexpected constraints encountered during the week.

1.2.3 Version Control

For collaborative management of code sources, Oracle Labs uses the Bitbucket platform for any interaction with code source.

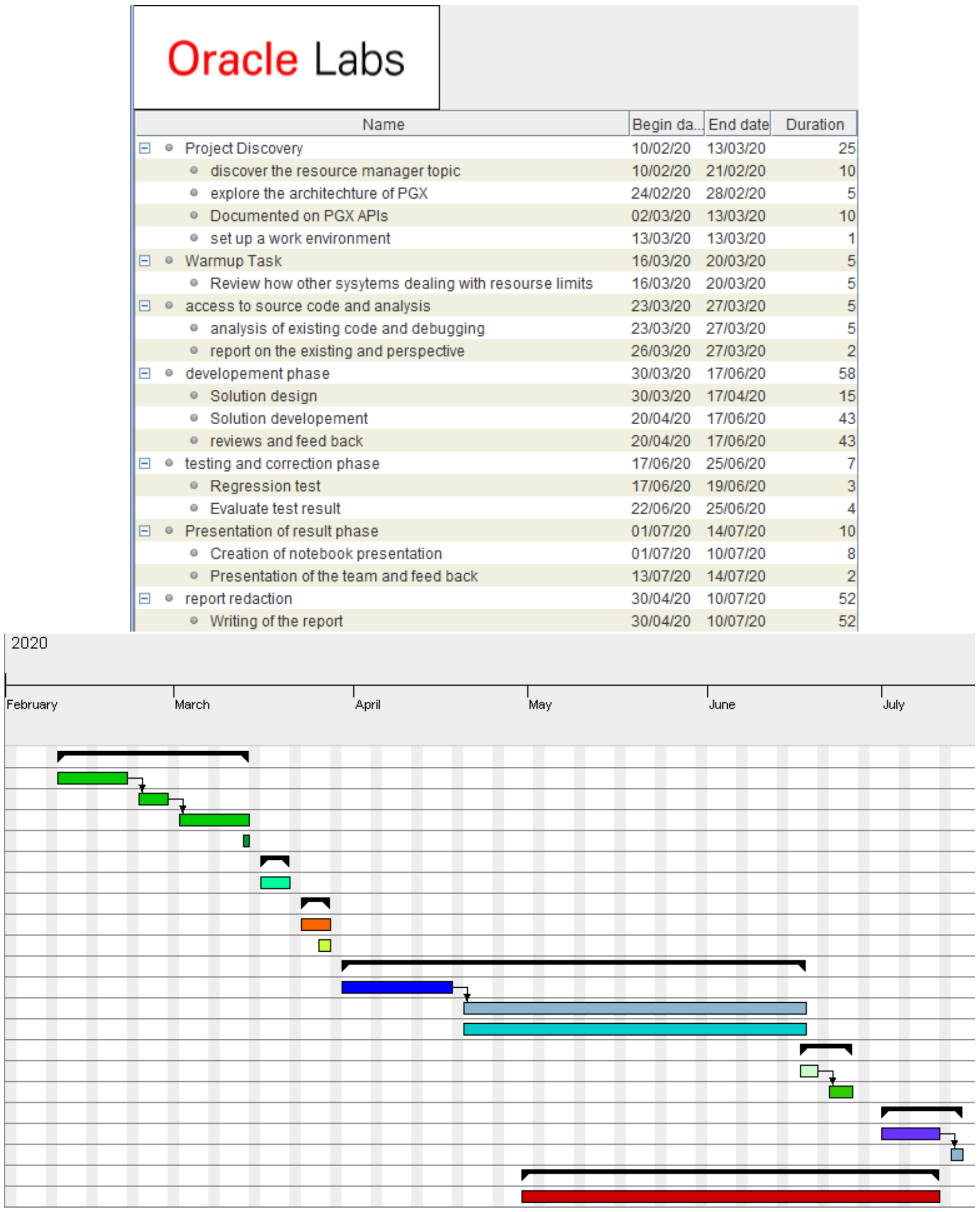
Bitbucket is our Git repository management solution designed for professional teams. It gives you a central place to manage git repositories, collaborate on your source code and guide you through the development flow. It provides awesome features that include:

- **Access control :** to restrict access to your source code.
- **Workflow control :** to enforce a project or team workflow.
- **Pull requests :** with in-line commenting for collaboration on code review.
- **Jira integration :** for ability to trace the development flow.

Confluence, Bitbucket and Jira platforms belong to the same software manufacturer ATLASSIAN. Tickets and code source are associated and the confluence pages have access to Jira labels in order to describe the new features and archiving the past ones.

1.2.4 Project planning

The following GANTT diagram illustrate the several running phases of the project :



Name	Begin da...	End date	Duration
Project Discovery	10/02/20	13/03/20	25
discover the resource manager topic	10/02/20	21/02/20	10
explore the architechture of PGX	24/02/20	28/02/20	5
Documented on PGX APIs	02/03/20	13/03/20	10
set up a work environment	13/03/20	13/03/20	1
Warmup Task	16/03/20	20/03/20	5
Review how other sysytems dealing with resourse limits	16/03/20	20/03/20	5
access to source code and analysis	23/03/20	27/03/20	5
analysis of existing code and debugging	23/03/20	27/03/20	5
report on the existing and perspective	26/03/20	27/03/20	2
developement phase	30/03/20	17/06/20	58
Solution design	30/03/20	17/04/20	15
Solution developement	20/04/20	17/06/20	43
reviews and feed back	20/04/20	17/06/20	43
testing and correction phase	17/06/20	25/06/20	7
Regression test	17/06/20	19/06/20	3
Evaluate test result	22/06/20	25/06/20	4
Presentation of result phase	01/07/20	14/07/20	10
Creation of notebook presentation	01/07/20	10/07/20	8
Presentation of the team and feed back	13/07/20	14/07/20	2
report redaction	30/04/20	10/07/20	52
Writing of the report	30/04/20	10/07/20	52

Figure 1.6 : – GANTT diagram.

Before starting the project, it is essential to plan for its implementation. The objective of the planning of a project is, on the one hand, the division of the latter into several intermediate phases to allow a better estimate of the total duration of the project and the necessary resources, and on the other hand the sequential validation with a view to ensure compliance with the needs expressed. Thanks to the meetings held with the supervisor,we were well informed on the different phases of the project as well as their progression in relation to time. This consisted of five phases as described in the course of the project.

1.2.5 Work delivery

The work delivery required during our project were either pieces of code sources (Pull Request),presentations or reports on the Confluence page.
This page is very useful for document all the tasks accomplished during a project and represent a real source of information for the rest of the team members. This therefore imposes a perfect combination between work done and documentation at page level so that the team is on the same picture.

The table below presents a list of deliverable produced during the realization of our project. We can notice that each phase of the realization is accompanied with one or more deliverable.

Table 1.2 - List of deliverable product.

Step	**Deliverable**
End of internship project	- Source code - Batch profile - Test book - Organic specification
Launch and planning	Project management and planning methodology
Study of requirements	Specification file
Design	design page
Evaluation	Test Scenarios Report
Final project	Pull request components

A 30-minute meeting with the PGX.SM team were held each week for a synchronization of the team's work and in particular discussing changes and prospects or potential deadlocks.

Conclusion

The first chapter presented the host organization and helped position the project in its general framework. Indeed, we were able to delimit the perimeter of our project as well as the expected objectives. The goal is to support a Resource Manager prototype on our PGX single machine engine. We are therefore led to monitor the completion of the various project tasks according to the adequate working method. The second chapter will focus on the definition of the technical environment of the project.

Chapter 2

Technical background and analysis

The purpose of this chapter is to present the technical context of the project.Therefore we will explain with more details the functional and technical architecture of the PGX prototype and modules, also at the same time, we will offer a detailed analysis of it, in order to understand its functioning and identify the Resource Manager feature on our engine.

***

2.1 Technical study

in this section, we discuss the technical study on PGX, we begin with supported models of graphs, then an architecture of PGX components, also we explain the server design of our single engine part, and finally we see the bench-marking results on different systems.

2.1.1 PGX graph models

Currently, PGX support two models of graphs:

2.1.1.1 Homogeneous graph model

PGX adopts the property graph model. In this model vertices and edges can be associated with a set of values, often called properties, where each one is identified with a unique name. For example, the small sample graph below has two properties for vertices ('name' and 'age') and one for edges ('relationship').

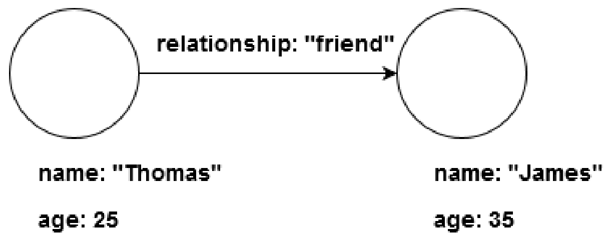


Figure 2.1 : – Homogeneous graph.

The PGX property graph model has the following characteristics:

• **Properties are typed:** Properties of vertices and edges are always typed and their type is declared when they are created or loaded.

• **Id is a special property:** vertex-id/edge-id is a special property that uniquely identifies the vertices/edges in the graph. However, it is optional for PGX graphs to have a vertex-id/edge-id.

2.1.1.2 Heterogeneous graph model

The PGX partitioned graph model is a representation of a property graph particularly suited for graphs having vertices and edges of different "types", where each type of vertex or edge has a different set of properties. For example, in a graph that represents Linkedin network, we will had people and profession, vertices of type "Person" would have properties such as "Name", while vertices of type "Profession" would have properties such as "Field". Similarly, edges from "Person" to "Person" may have properties like "follows", while edges from "Person" to "Profession" may have properties such as "is a".there's an exemple on the graph below:

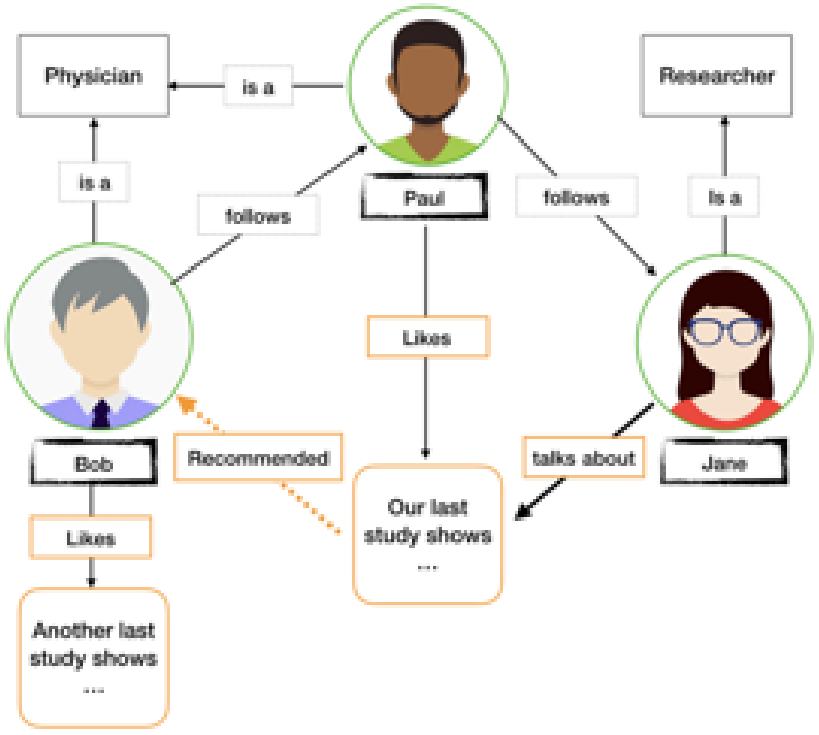


Figure 2.2 : – Heterogeneous graph.

The PGX property graph model has the following characteristics:

• **Properties are typed:** Properties of vertices and edges are always typed and their type is declared when they are created or loaded.

• **Id is a special property:** each table has his own keys, that why we add a table name for each type of vertices or edges to distinguish the conflicts of having the same Id.

- Deduction

Since the homogeneous graphs are such a special case for the heterogeneous type, as future plans,the PGX team will try to merge those models on global one (the heterogeneous model).

2.1.2 PGX architecture

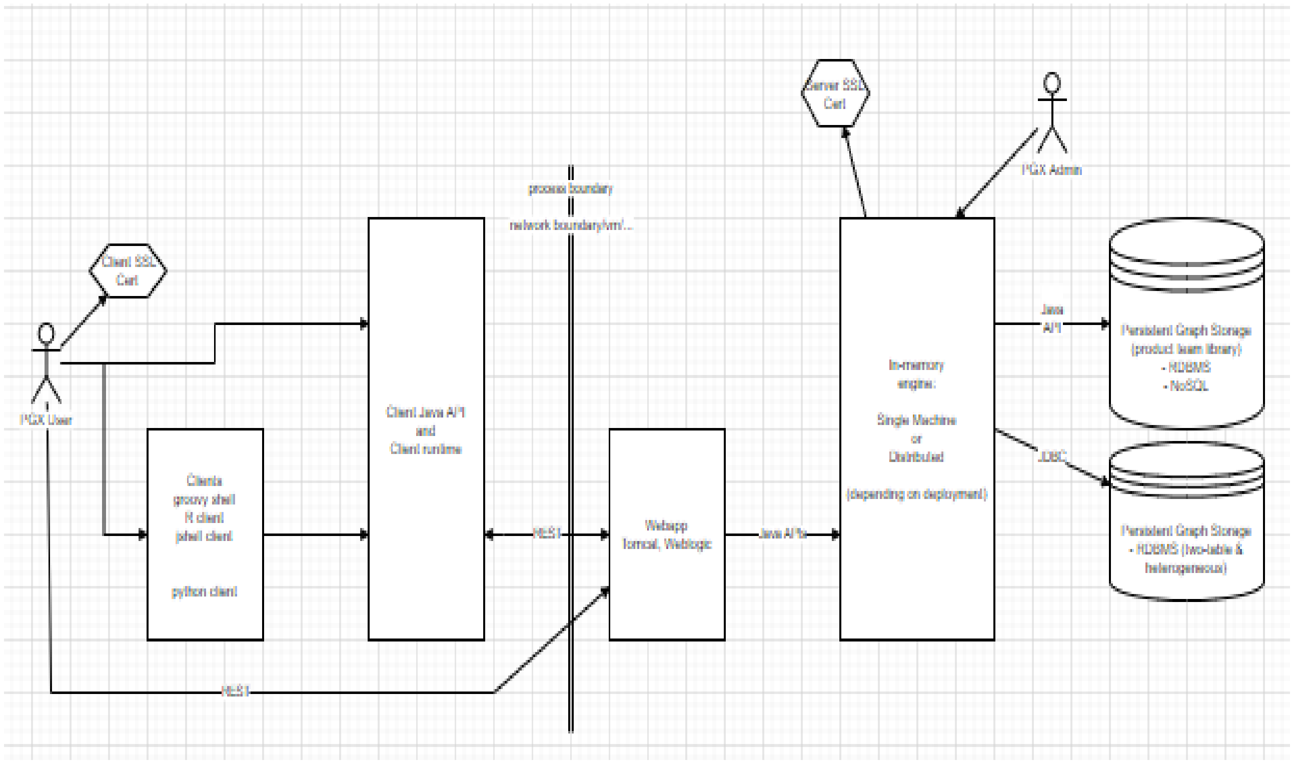

Server SSL Cert
PGX Admin
process boundary
network boundary/vm/...
Client SSL Cert
PGX User
Client Java API and Client runtime
Clients groovy shell R client jshell client python client
REST
Webapp Tomcat, Weblogic
Java APIs
In-memory engine: Single Machine or Distributed (depending on deployment)
Java API
JDBC
Persistent Graph Storage (product team library) - RDBMS - NoSQL
Persistent Graph Storage - RDBMS (two-table & heterogeneous)
REST

Figure 2.3 : – PGX architecture.

Now, let explain the components of the architecture:

- **clients :** A set of clients that can reach the Client Java APIs directly through the shell (e.g. groovysh, jshell).
- **Client Java API + Runtime :** Java APIs exposing the PGX functionality to the user.
- **Webapp :** A layer that takes REST requests and translates them back into Core API requests.
- **In-memory engine :** Contains functionality for Loading or storing graphs from various persistent storage providers into memory or from memory into persistent,Run graph algorithms on graphs in memory, and Run queries on graphs in memory.
- **Persistent Graph Storage :** We connect to Oracle RDBMS and load the persistent graph from it either by Local server filesystem, HTTP(S) servers, FTP(S) servers ...

  -As last note, the PGX admin had access to the memory-engine.

2.1.3 PGX server design

PGX is designed for a Server-Client usage model.That can be used for the following situations:

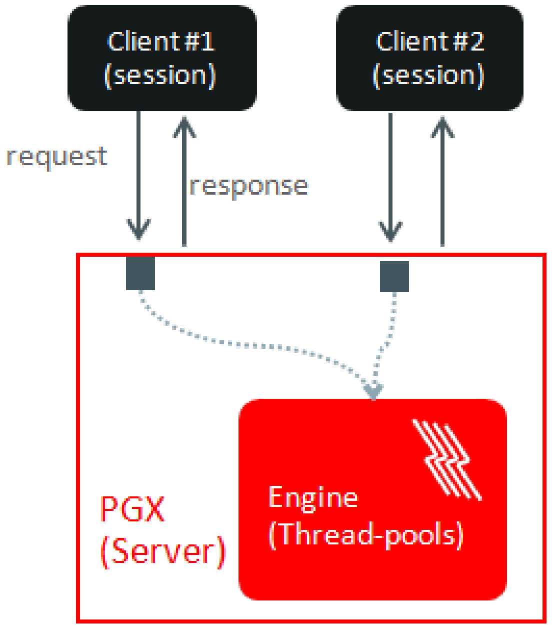


Figure 2.4 : – PGX server Design.

- The PGX server is initiated on a server-class machine, while PGX clients (remotely) connect to the PGX server; multiple clients can connect to the same PGX server at the same time.

- Each PGX client sends out its requests to the PGX server, and the server returns responses for them. However, these requests are processed by the PGX server asynchronously, they are queued up first and then processed later, when the resources are available. The client can poll the server to check if a request has been finished.

2.1.4 PGX memory management

In this section, we will see how the PGX project manage memory before we support the new Resource Manager prototype.

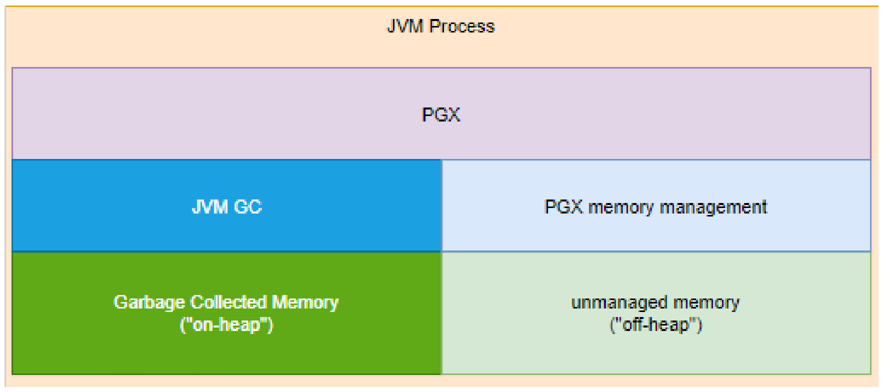

JVM Process
PGX
JVM GC
PGX memory management
Garbage Collected Memory ("on-heap")
unmanaged memory ("off-heap")

Figure 2.5 : – PGX memory management.

- **Explanation**

Although PGX is mainly a Java application, it stores some of its graph data in off-heap memory, meaning in memory locations outside the control of the Java virtual machine. This is because of Java's 32bit array-length limitation. PGX uses off-heap memory to store all vertices, edges and properties, and as big part of the project we need to manage the off heap memory and specially is the one that contains the huge objects allocations.

2.2 analysis study

The study of the existing allows us to determine the weak and the strong points of a current product in order to be able to determine the needs of the customer, with a view to taking them on consideration when designing and implementing the solution. This is why we conduct a benchmarking process on the most known data bases and their process of managing resources, by comparing those features, we can come up with an optimal solution to our problematic, and also an analysis study on the GC concept and the technical approach we consider for the introduced APIs.

2.2.1 the general need of the project

The main reason behind the need of resource manager process on every system, is that we can run more services on single engine witch improve the server availability, and organize the flow of our users on the server.

PGX currently has (basic) task cancelling, and tracks how much memory tasks are allocating, but we don't have a resource manager that will look at per task/per session memory consumption and kill tasks if they go over some limit.

The main criteria of the new model:

- As first step, we review how other systems (e.g., the DB) handle resource limits.
- The critical resource we want to track is the memory.
- we define how we are going to enforce the limits (e.g, kill session, forcefully destroy resources, ...).
- PGX handle the resource utilization and cancel tasks that go over the specified limits.
- We need to treat limits of resources as soft limits and we enforce them one we run out of resources.
- We still have the option to enforce hard limits.
- Provide PGX with an admin-controllable way to trigger graph GC.
- Force the GC if we are about to run out of memory or even trigger at the needed time.

2.2.2 the non-functional needs of the project

The non-functional specifications are requirements that do not directly relate to the specific services provided by the system. They may relate to properties and constraints fulfilled by the entire system.

The main non-functional needs of our project are summarized in the following table:

Table 2.1 - List of non-functional needs.

Requirements	Description
Reliability	The prototype must work without failure and execute the expected functions with the required precision
Availability	The prototype must be available each time the client need it
Maintainability	Makes the prototype easier to locate and correct the bugs on the future
Rapidity	The prototype must execute the requested features with a reasonable time
Scalability	Should be possible to extend the prototype on the next version
Security	the prototype must respect access controls, which prevent unauthorized personnel from entering or accessing a system, protecting the server information by SSL/TLS Security Certificates
Technical constraints	The access to RDBMS should be fast and flexible for the clients

2.2.3 Benchmarking

On this warm-up task, I was assigned to run a benchmarking study on different data bases systems, and how they handle the resource limits, on this study I checked different systems and their decisions of tackling the problem as we can listed below :

* **Mongo db :** They set a quota of 16 Mb on document and 100 Mb on RAM per user.
* **Couch db :** They had restrictions on number of active users (max_users = 100) combined with the restriction of files per each user (max_files_per_user = 1000).
* **HBase :** They set a quota on requests per user of 100 req/min .
* **Neo4j :** They set a quota on threads per user of 400 threads/5min .
* **Oracle DB :** They set an active session pool consists of specified maximum number of user sessions allowed to be concurrently active within a group of users.

Conclusion

Finally, we decide to use the Oracle db approach since it's adopt the concurrent manner between users for managing the resources.

2.2.4 Server behavior study

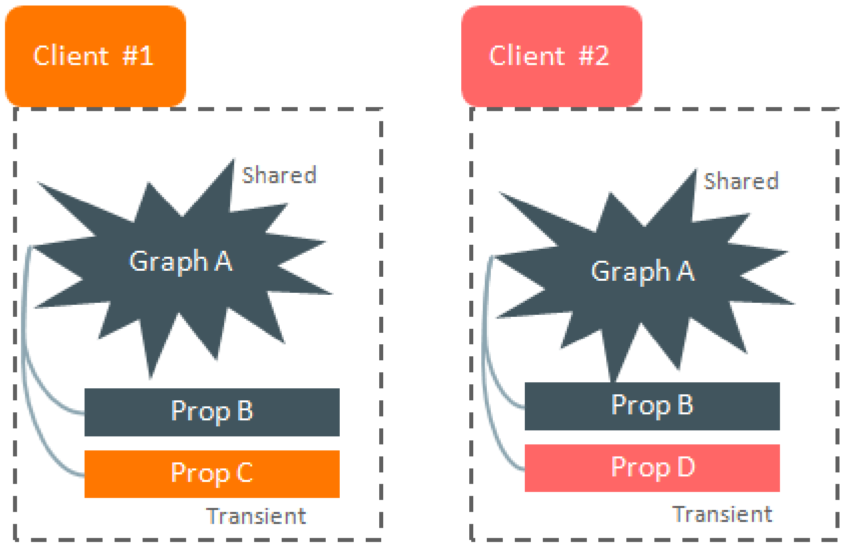


Figure 2.6 : – Server concurrent client.

PGX supports data isolation between concurrent clients. Conceptually, each PGX client has its own private workspace, which we call a 'session'. Sessions are isolated. that is, a client cannot access any data from another client's session.

Note that in PGX the loaded graph instance and its original properties are immutable, which makes it trivial to share a graph instance among multiple clients. However, each client can dynamically add additional vertex or edge properties to a loaded graph. Such a transient property is private to each session and thus is not visible to other sessions. Similarly, each client can create a mutated version of the loaded graph, in which case the PGX server creates a private graph instance for the client.

2.2.5 The Garbage Collector Concept

Lisp is especially notable as both the first functional programming language and the first language to introduce garbage collection, then Java adopt the concept and upgrade it by bench of new algorithms to manage memory allocation.

The advantages that garbage collected languages offer to software development are legion. It eliminates whole classes of bugs, such as attempting to follow dangling pointers that still refer to memory that has been reclaimed or worse, reused in another context. It is no longer possible to free memory that has already been freed. It reduces the chances of programs leaking memory, although it cannot cure all errors of this kind. It greatly simplifies the construction and use of concurrent data structures. Above all, the abstraction offered by garbage collection provides for better software engineering practice. It simplifies user interfaces and leads to code that is easier to understand and to maintain, and hence more reliable. By removing memory management worries from interfaces, it leads to code that is easier to reuse.

2.2.6 The Mark and sweep Approach

Mark and Sweep algorithms and the one that we adopt on our GC model use conceptually the simplest approach to garbage by such ignoring such objects. What this means is that after the marking phase has completed all space occupied by the non visited objects is considered free and can thus be reused to allocate new objects.

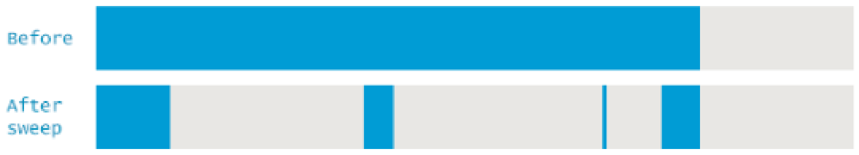


Figure 2.7 : – Mark and Sweep process.

- **Technical Explanation**

The Mark and Sweep algorithm is a straightforward embodiment of the recursive definition of pointer reachability. Collection operates in two phases. First, the collector traverses the graph of objects, starting from the roots (registers, thread stacks, global variables) through which the program might immediately access objects and then following pointers and marking each object that it finds. Such a traversal is called tracing. In the second, sweeping phase, the collector examines every object in the heap: any unmarked object is deemed to be garbage and its space reclaimed.

Conclusion

This chapter was dedicated for the technical and analysis study of the resource manager prototype, according a details on various components of PGX, on one hand the technical study was about the graph models, general architecture, server design, and finally the memory management of our project, on the other hand, we analyse the general need of the project, benchmarking between existing systems, then our server behavior and their limits, finally a garbage collector study and the use cases of our model.

This phase is essential and gather a huge importance on tackling the project sides. This chapter will be the basis for the design and implementation part of the proposed solution.

Chapter 3

Design and modeling

This chapter will present the design and implementation phases of the solution that resolve the problem we encounter during the analysis part. Among other things, it will include the UML diagrams for detecting the global flow of the project and the test environment, according to the different processing steps of the initial prototype.

***

3.1 Design and implementation steps

in this section, we discuss the new design and implementation steps to solve the problem, I divided on two parts, one for enforcing the resource limits and other for the GC feature.

3.1.1 Enforce the resources limits

On this part, we discuss the taken steps of design to handle the resource limits, including the process of tracking the memory on our server to check that our limits works fine, the use cases for both the admin and the user, and a sequence diagram for the user-system interaction.

3.1.1.1 Memory design

As a first step of managing the memory resource, we come up with the new design as I explain on the figure below :

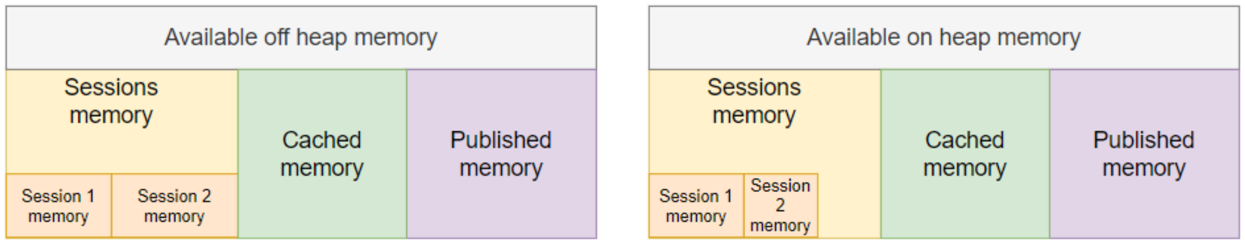


Figure 3.1 : – Memory design.

On those two designs, we limit the memory for the on-heap memory and the off-heap memory, let's explain the divided sections that we have :

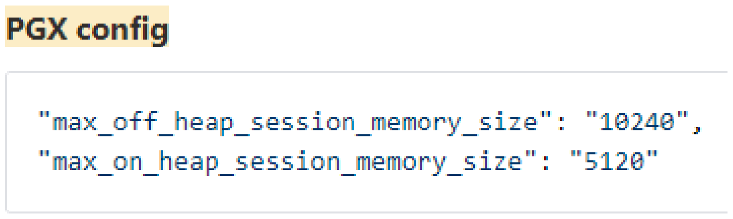
- **Sessions Memory :** contains the users sessions private graphs, with their properties and result sets from the engine interaction.

- **Cached Memory :** contains the cached graphs, that means the loaded graphs from a source (RDBMS or File System) .

- **Published Memory :** contains the published graphs and their properties.

3.1.1.2 configuration model

After several discussions, we decide to set a limit per a session as it's the workflow for the users. We need to add those new limits on both on-heap memory and off-heap memory, and by default the total amount of memory PGX is allowed to allocate is twice the default maximum on-heap size.


PGX config

```
"max_off_heap_session_memory_size": "10240",
"max_on_heap_session_memory_size": "5120"
```

Figure 3.2 : – Session configuration.

Above configuration has following effects :

- Each session cannot use more than 10 GB to store off-heap private data (results sets and all intermediate used memory during query evaluation)
- Each session cannot use more than 5 GB to store on-heap private data

For those new configurations, they are the set up of our workflow on managing the memory, for the on heap memory it's already taken by JVM GC, but the off heap memory still not managed .

3.1.1.3 Memory tracking

The next step on the model is to find a way for tracking the memory, based on the DS that we're handling on the project, the bottleneck of the problem was the tremendous numbers of APIs that PGX framework support, so as a first version of limiting the memory resources we will only consider the PGQL APIs that we listed on the table below:

Table 3.1 - List of PGQL APIs.

API	Usage
queryPgql	Submits a pattern matching select only query
queryPgql	Submits a pattern matching select only query
executePgql	Submits a pattern matching query (from a PgxGraph). Supports both select and modify queries
executeAnyPreparedStatement	Submits a select query
getPgqlResultSet	Gets a query result set by ID
createFrameFromPgqlResult	Copies the specified result set into a new PgxFrame
destroyPgqlResultSet	Destroys a pattern matching result set that is bound to a session
explainPgql	Explain the execution plan of a pattern matching query

3.1.1.4 Class diagram for allocations

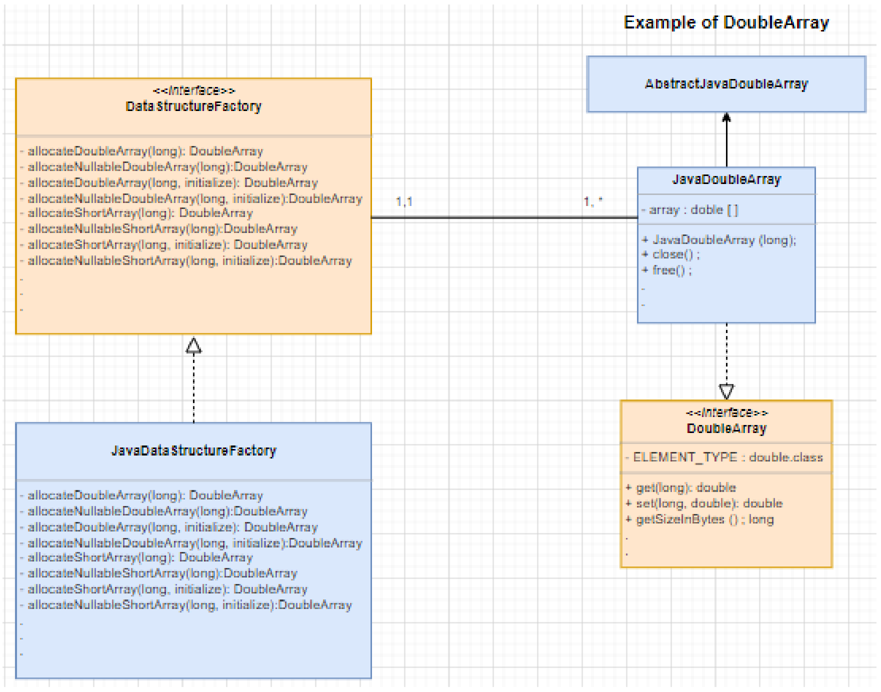

Example of DoubleArray
<<Interface>>
DataStructureFactory
- allocateDoubleArray(long): DoubleArray
- allocateNullableDoubleArray(long):DoubleArray
- allocateDoubleArray(long, initialize): DoubleArray
- allocateNullableDoubleArray(long, initialize):DoubleArray
- allocateShortArray(long): DoubleArray
- allocateNullableShortArray(long):DoubleArray
- allocateShortArray(long, initialize): DoubleArray
- allocateNullableShortArray(long, initialize):DoubleArray
1,1
1, *
AbstractJavaDoubleArray
JavaDoubleArray
- array : doble [ ]
+ JavaDoubleArray (long);
+ close() ;
+ free() ;
JavaDataStructureFactory
- allocateDoubleArray(long): DoubleArray
- allocateNullableDoubleArray(long):DoubleArray
- allocateDoubleArray(long, initialize): DoubleArray
- allocateNullableDoubleArray(long, initialize):DoubleArray
- allocateShortArray(long): DoubleArray
- allocateNullableShortArray(long):DoubleArray
- allocateShortArray(long, initialize): DoubleArray
- allocateNullableShortArray(long, initialize):DoubleArray
<<Interface>>
DoubleArray
- ELEMENT_TYPE : double.class
+ get(long): double
+ set(long, double): double
+ getSizeInBytes () ; long

Figure 3.3 : – Class diagram for tracking Double Array allocation.

The aboved diagram showed the several data structures on the DataStructureFactory interface that we used to allocate of the memory for graphs edges properties and vertices properties.

On the class diagram, I detailed just the case of the double array and the same for others types, technically speaking, we can track the memory of the double array by getting the size on bytes of the array once we need it.

3.1.1.5 Use cases for memory limits

On this section, we will discuss the different use cases that will be included by the Resource limits prototype either for a normal or an admin user.

The below use case diagram explain the interaction between the user and the shared memory engine :

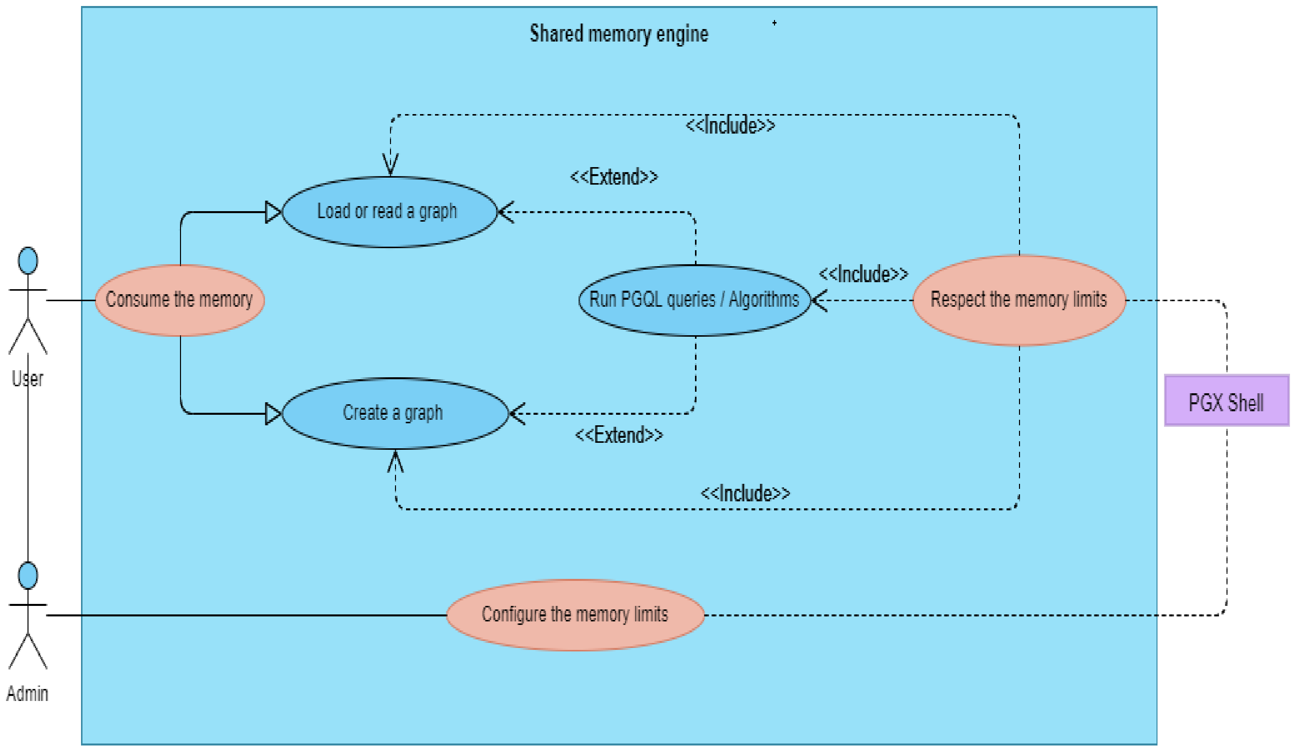


Figure 3.4 : – Use case diagram for memory limits.

For the two cases below, it's about supporting new limits on the memory configuration by the Admin, and checking the well functioning of the PGQL queries .

Table 3.2 - Use case for normal user resource limits.

Actor	User
System	Shared memory Engine
Precondition	Connected to the shell work space
Story	We need to be able to run PGQL query if it respects my quota

Table 3.3 - Use case for admin configuration.

Actor	Admin
System	Shared memory Engine
Precondition	Connected to the shell work space
Story	We need to be able to configure memory quota limits for user / session

3.1.1.6 Sequence diagram for memory limits

The below sequence diagram explain the interaction between the user and the shared memory engine that got displayed on the java / groovy Shell :

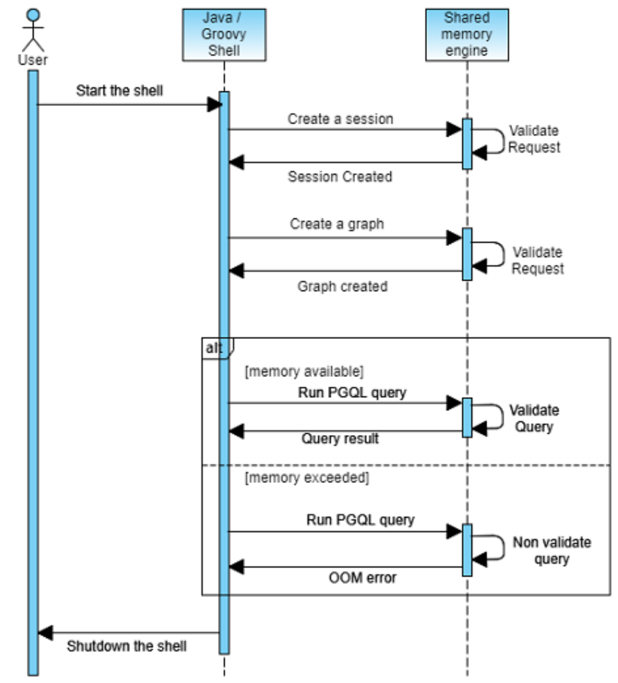


Figure 3.5 : – Sequence diagram for memory limits.

3.1.2 The GC feature Design

On this part, we discuss the several steps of design to handle the GC approach for the off heap memory of the server, accompany with a simple analogy and illustration of the logic we covered, the use cases for both the admin and the user, and the sequence diagrams for the user-system interaction.

3.1.2.1 The internal architecture of PGX

The following figure depicts the overall internal architecture design of PGX :

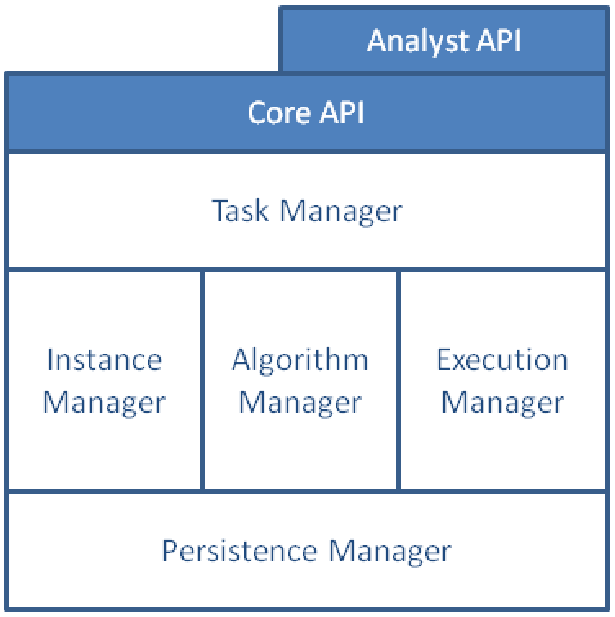


Figure 3.6 : – The internal architecture of PGX.

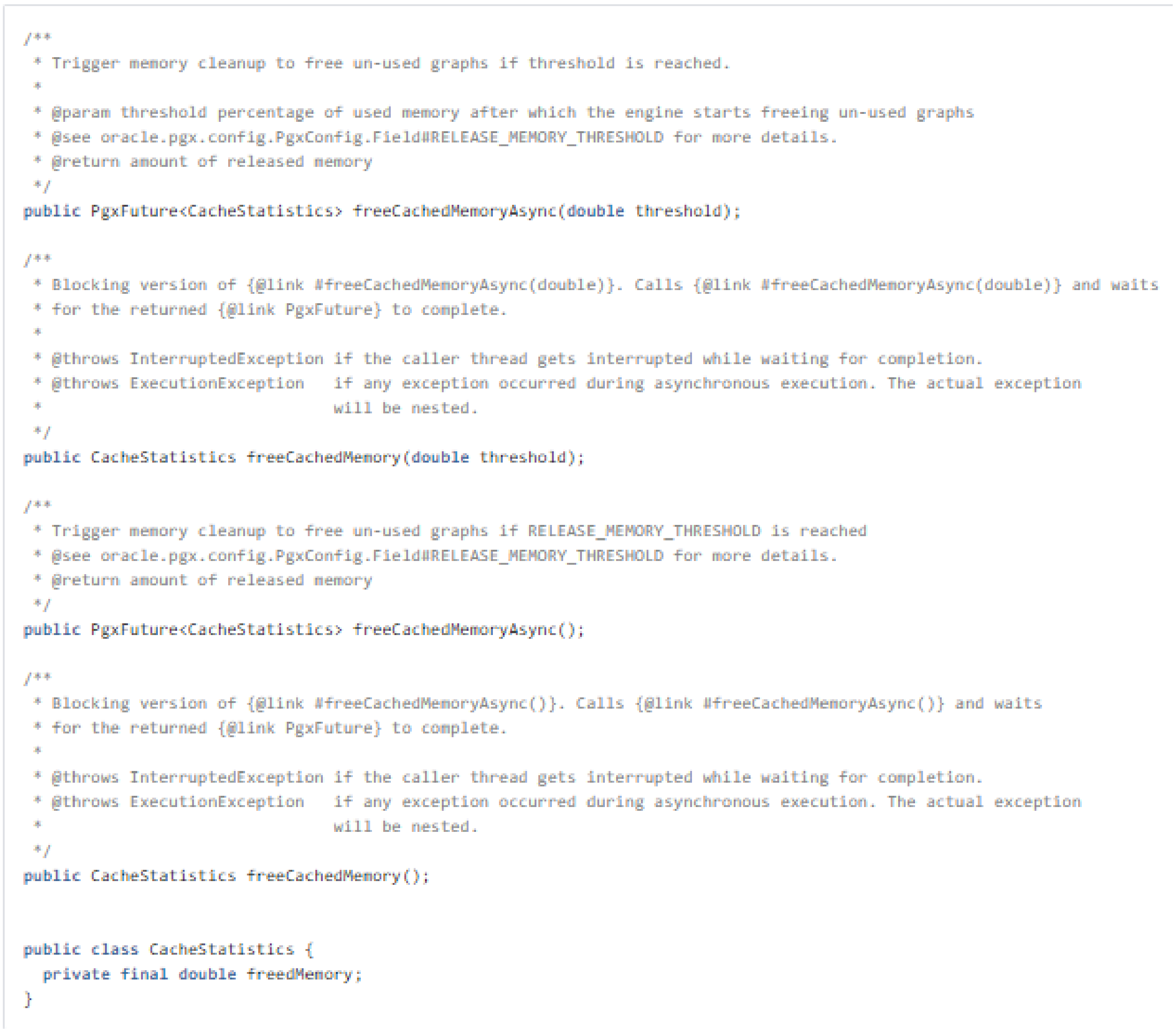
On this big picture of the architecture, we're only interested on two parts for implementing the new APIs model :

- The Instance Manager takes care of all the in-memory instances: graphs, properties, and collections. Specifically, it manages which graphs are visible to which clients, how they are named and what their types are.
- The Persistence Manager is in charge of loading the graph data from a storage layer (i.e. file system).

3.1.2.2 The Admin APIs model

Since our PGX project need to handle the off heap memory, we come up with new APIs to hold the required job.

There's a simple signature functions of the holding APIs :



```
/**
 * Trigger memory cleanup to free un-used graphs if threshold is reached.
 *
 * @param threshold percentage of used memory after which the engine starts freeing un-used graphs
 * @see oracle.pgx.config.PgxConfig.Field#RELEASE_MEMORY_THRESHOLD for more details.
 * @return amount of released memory
 */
public PgxFuture<CacheStatistics> freeCachedMemoryAsync(double threshold);

/**
 * Blocking version of {@link #freeCachedMemoryAsync(double)}. Calls {@link #freeCachedMemoryAsync(double)} and waits
 * for the returned {@link PgxFuture} to complete.
 *
 * @throws InterruptedException if the caller thread gets interrupted while waiting for completion.
 * @throws ExecutionException   if any exception occurred during asynchronous execution. The actual exception
 *                              will be nested.
 */
public CacheStatistics freeCachedMemory(double threshold);

/**
 * Trigger memory cleanup to free un-used graphs if RELEASE_MEMORY_THRESHOLD is reached
 * @see oracle.pgx.config.PgxConfig.Field#RELEASE_MEMORY_THRESHOLD for more details.
 * @return amount of released memory
 */
public PgxFuture<CacheStatistics> freeCachedMemoryAsync();

/**
 * Blocking version of {@link #freeCachedMemoryAsync()}. Calls {@link #freeCachedMemoryAsync()} and waits
 * for the returned {@link PgxFuture} to complete.
 *
 * @throws InterruptedException if the caller thread gets interrupted while waiting for completion.
 * @throws ExecutionException   if any exception occurred during asynchronous execution. The actual exception
 *                              will be nested.
 */
public CacheStatistics freeCachedMemory();


public class CacheStatistics {
  private final double freedMemory;
}
```

Figure 3.7 : – APIs signatures and descriptions.

All methods in the Analyst API support asynchronous execution.
This is a design choice, since the execution of a graph algorithm can take a long time if the size of the graph is large. The application thread invokes the Analyst API method. It may continue to perform other operations while the algorithm is running. If the application thread decides to wait for the result to be available, it can invoke the blocking version directly (e.g. call freeCachedMemory() instead of freeCachedMemoryAsync()), which internally calls the get() method on the PgxFuture object.

We add the CachedStatistics class on our model to support the needed memory statistics, on our case, we only take care of the memory freed after the GC process.

Note: All PGX APIs follow the same design approach of supporting both asynchronous and synchronous invocation.

3.1.2.3 PGX and JVM analogy

There's the PGX analogy of the supported GC on the table below :

Table 3.4 - JVM and PGX analogy

JVM	**PGX**
works dynamically	works periodically
GC algorithms	memoryCleanup logic
Runtime.getRuntime().gc()	instance.freeCachedMemory() / .freeCachedMemory(threshold)

• JVM :

Generally speaking, the JVM works by performing garbage collection to the on-heap memory when it needs more memory to continue execution, but we can force it by calling gc() through the Runtime class.

• PGX :

From the off heap memory side, we had the memoryCleanup logic that get invoked peiodically on the server, with the option to force the call by the freeCachedMemory() or freeCachedMemory(threshold) but only for a PGX admin.

3.1.2.4 Use cases for GC APIs

On this section, we will discuss the different use cases that will be included by the GC APIs either for a normal or an admin user.

The below use case diagram explain the interaction between the user and the shared memory engine :

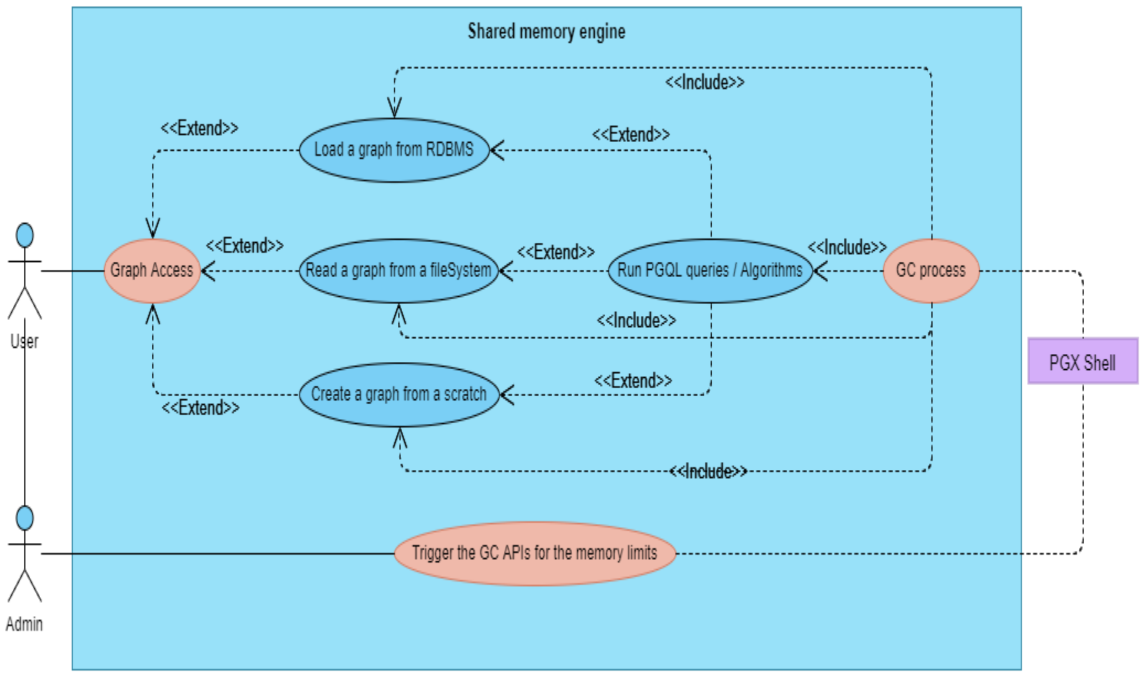


Figure 3.6 : – Use case diagram for GC APIs.

For the two cases below, it's about the GC service for the users, and also the admin APIs to force it during the execution .

Table 3.5 - Use case for normal user GC service .

Actor	User
System	Shared memory Engine
Precondition	Connected to the shell work space
Story	We offer for users the GC service periodically

Table 3.6 - Use case for admin GC APIs.

Actor	Admin
System	Shared memory Engine
Precondition	Connected to the shell work space
Story	We can trigger the graph GC to free the off-heap memory

3.1.2.5 the logic of the memoryCleanup

The below activity diagram shows the main process of the memoryCleanup logic :

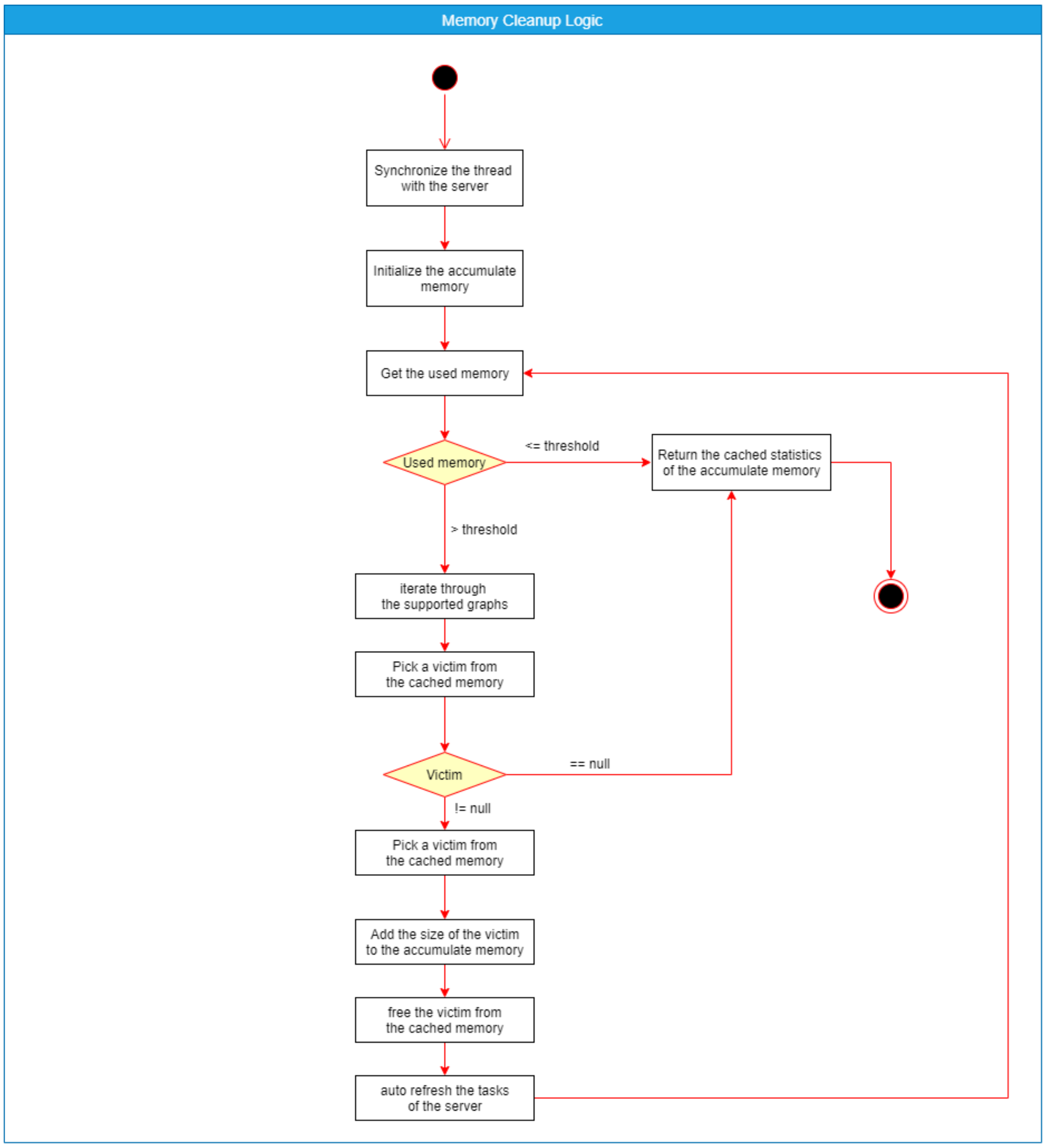

Memory Cleanup Logic
Synchronize the thread with the server
Initialize the accumulate memory
Get the used memory
Used memory
<= threshold
Return the cached statistics of the accumulate memory
> threshold
iterate through the supported graphs
Pick a victim from the cached memory
Victim
== null
!= null
Pick a victim from the cached memory
Add the size of the victim to the accumulate memory
free the victim from the cached memory
auto refresh the tasks of the server

Figure 3.7 : – Activity diagram ≪ memoryCleanup process ≫.

Illustration :

∗ **Thread Synchronization :**

The memory Cleanup logic begin by synchronizing the current thread with the main server thread, technically, we configure the run() method of the main server thread to be invoked the current thread by a memory cleanup interval, for the standard use we set the interval on 10 min.

∗ **Infinite loop through the server tasks :**

after we initialize the accumulate memory with 0 bytes, we begin the infinite loop through the registered tasks on the server state API, and each time we pick a victim, we add the size of it to the accumulate memory and free it from the cached memory. technically, we pick a victim by checking that no available references are pointing to the processed graph.

∗ **The breaks conditions :**

As the memory cleanup interval, we got the release threshold that's responsible on deciding whether we need to free the memory from the server, or we still got the capacity to support more graphs, and also the base case scenario, when there is no more victims available we left the loop.

∗ **The return result**

Finally, once we finished freeing the non referenced graphs from the cached memory, and accumulating the size of those graphs on the accumulate memory, we converted it to the mega bytes and store it on the cached statistics field (freedMemory) as a final result of the algorithm.

3.1.2.6 Sequence diagrams

The sequence diagram below explain the interaction between a normal user and the GC service on the shared memory engine :

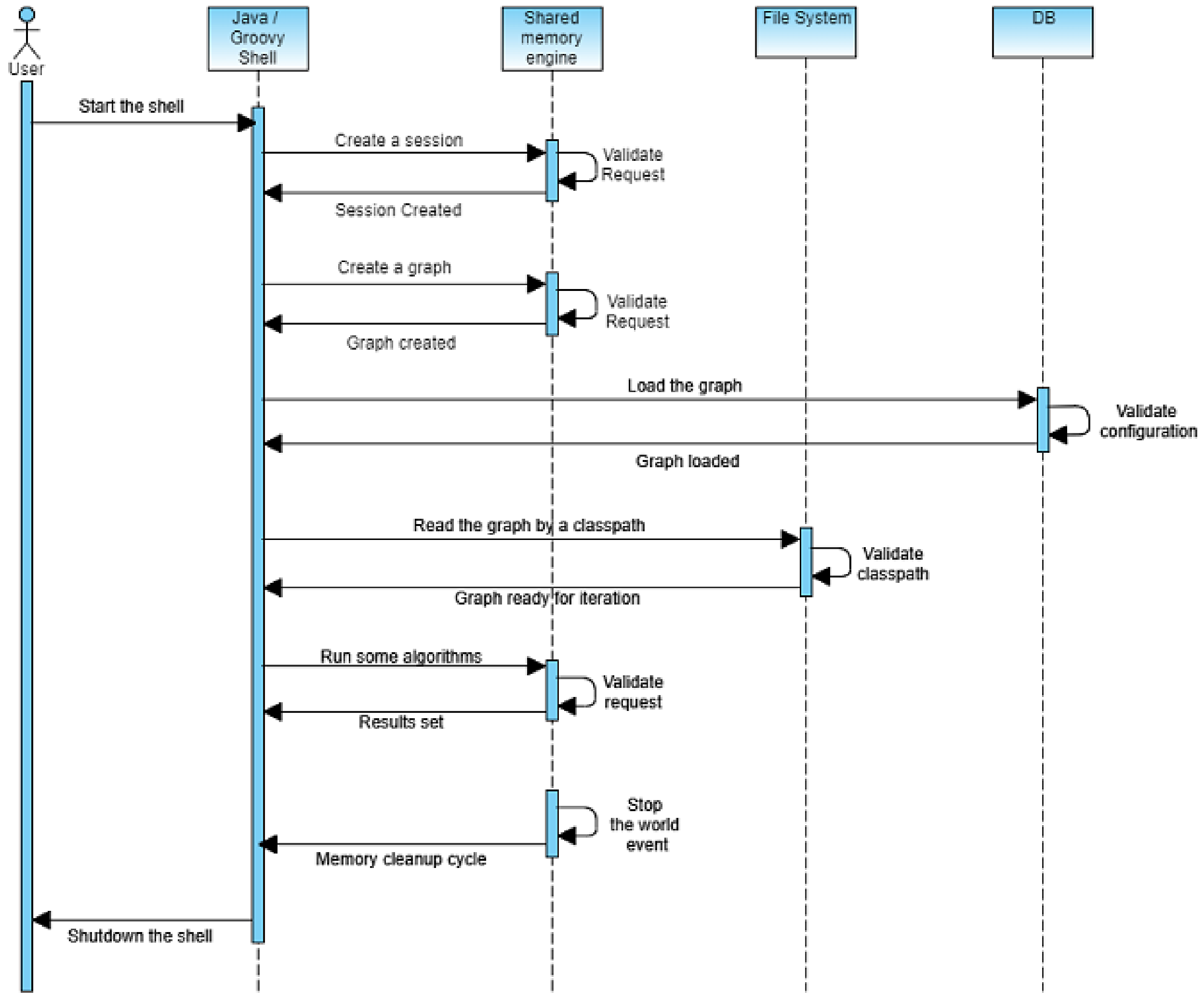


Figure 3.8 : – Sequence diagram ≪ memoryCleanup for normal user ≫.

The sequence diagram below explain how the admin can invoke the GC APIs service on the shared memory engine :

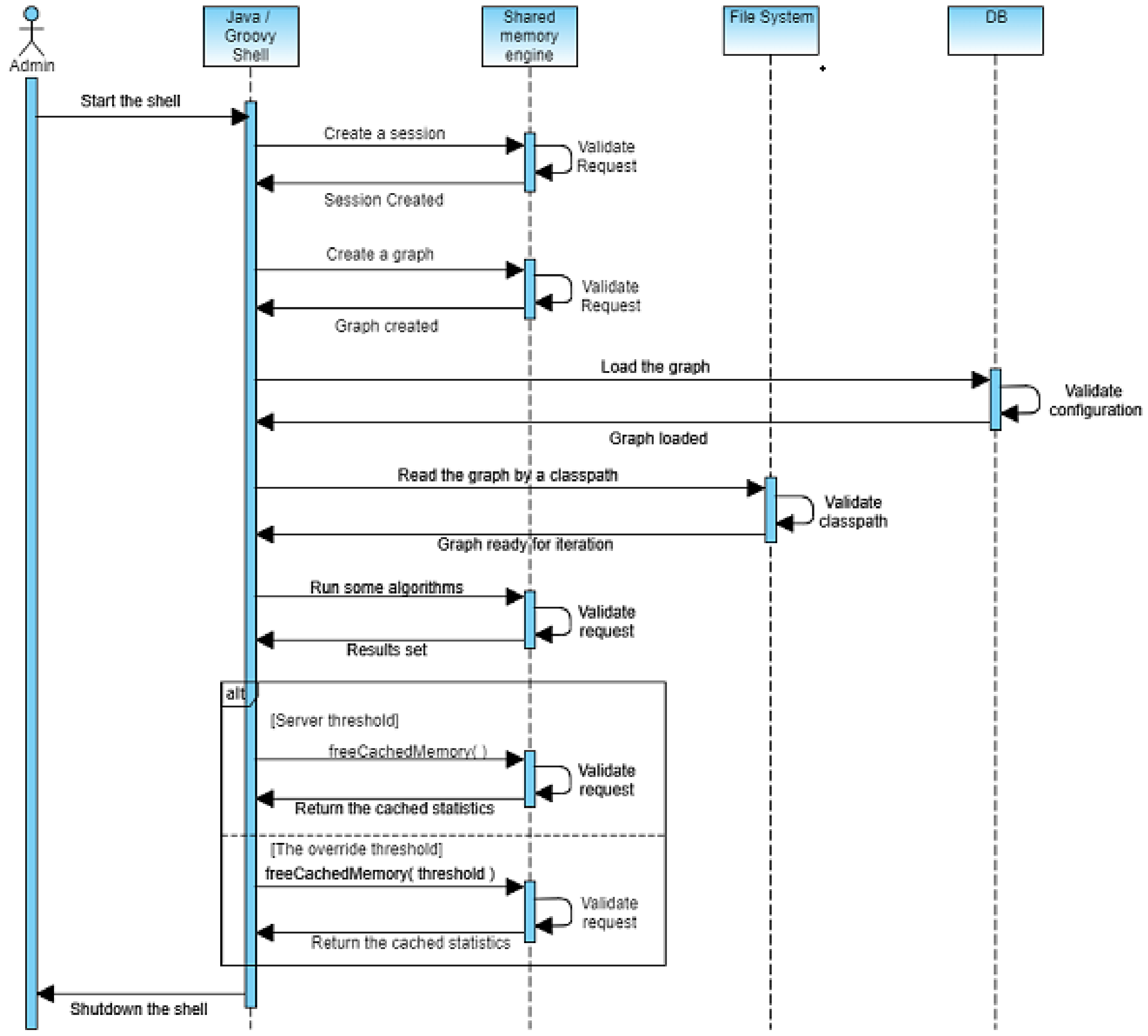


Figure 3.9 : – Sequence diagram « GC APIs for Admin ».

3.2 technical model of the solution

In this section, we illustrate an architecture of the supported solution, then we explain the technical constraints, the respected criteria and standards for achieving the final product.

3.2.1 Technical architecture of the solution

The schema below model the supported solution on the PGX framework architecture :

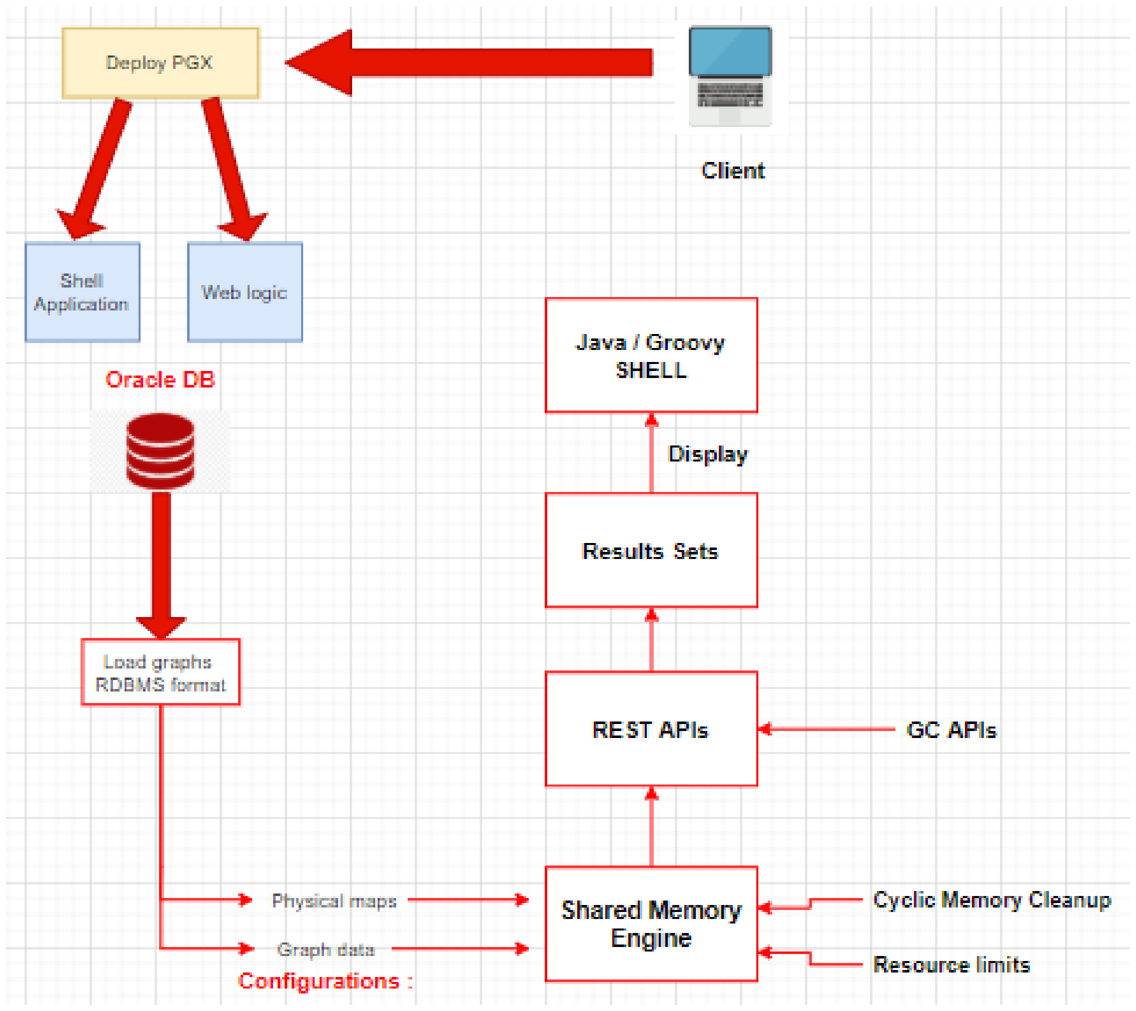


Figure 3.10 : – Technical architecture of the solution.

3.2.2 Technical standards and acceptance criteria

Including a new solution to an existing project need to respect several criteria and standards, we list the general aspects that we handled during the support of the solution :

∗ **Project structure :**

  As a first step to every existing project, we begin the journey by getting familiar to the code structure and exploring the different packages supported by PGX, then we begin to dig more on details on the API part to hold our solution.

∗ **Clean style :**

  Every single piece of code we add to the project need to be clean and well documented, for the clean code part we had that gradle task for checking the style to keep us on the good track and for the documentation part we had to respect the function policy and the utility from supporting the recent feature.

∗ **Prioritize the exist features:**

  During the implementation process, we should always prioritizing the use of the supported features beside begin from the scratch, using that strategy add a consistent source code to the project and keep it from including the same utile code twice.

∗ **Respect the third-party model**

  on the PGX third-party we use Gradle tool to include external binaries or other library modules to the build as dependencies, and since we need to introduce new code to the project, we must had a strong reason to add any dependency or manage to remove it, So it's easier for us to respect the balanced manner of our dependencies rules than introduce new ones.

∗ **Testing everything**

  Before we include any new features to the project, we go through several testing process that include Unit tests, integration tests or stress tests to make sure the expected results match the requirements needed form the feature, and finally we launch a regression test to confirm that the recent code does not affect the existing features on the project.

3.3 Test Environment

The work includes new source code to an existing project, to be sure that our modifications did not cause regressions, it's necessary to create a regression test environment, Among the most used and most powerful in terms of framework dedicated for testing on java, we found the **JUnit** framework and it's the one dedicated for testing on the PGX project.

JUnit

Figure 3.11 : – JUnit logo.

We use the JUnit framework based on the listed purposes :

- **Annotations and Assertions :**

  Provides various types of annotations to identify test methods, and also different types of assertions to verify the results of test case execution.

- **Fixtures Function :**

  Ability to provide a fixed base on which tests can be performed reliably and repeatedly. For our case this base will allow avoid repeated loading of configurations at the prototype level.

- **Simple use and redaction :**

  Writing tests in a compact way make them easy to understand without using logs or debugger, and it's open for you the possibility to track the tests separately.

- **Simple results :**

  Results well presented and provide all the information required .

Conclusion

This chapter was dedicated for the design and modeling of the resource manager prototype, we begin by introducing the design and implementation steps for the both main tasks, enforce the resource limits and the GC feature design, then we model our prototype on technical architecture and the technical standards with their acceptance criteria, finally the test environment used on the project.

This phase is essential to assimilate the core of the prototype and the taken steps during the development phase.

The next step, we begin by introducing the used technologies during the project, then we illustrate the work done through a practical demonstration aimed at valuing and giving meaning to improvements, as well as the other unrelated tasks that we support on our journey.

Chapter 4

Realisation and demonstration

This chapter is dedicated to all tools and technologies used to carry out the project, the practical presentation of the graphs on PGX, then the several steps of the delivery of the new product, also we illustrate a practical demonstration on the GC feature step by step, and finally the unrelated task that I gather simultaneously during the process.

***

4.1 Used technologies

The resource manager prototype as well as the solution we have introduced, is based on stack of rich technologies and tools which we present the main components on building our solution.

4.1.1 java language

Java is a general-purpose programming language that is class-based, object-oriented, and designed to have as few implementation dependencies as possible. It is intended to let application developers write once, run anywhere (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture.

Figure 4.1 : – Java logo.

on my part of PGX project we use the java core language part (standard edition) and specially the Stream APIs to process the objects collection and the completableFuture for the synchronisation and interaction between methods and features results.

4.1.2 Gradle

Gradle is an open-source build automation tool that is designed to be flexible enough to build almost any type of software.

Figure 4.2 : – Gradle logo.

Gradle avoids unnecessary work by only running the tasks that need to run because their inputs or outputs have changed. You can also use a build cache to enable the reuse of task outputs from previous runs or even from a different machine (with a shared build cache). Gradle runs on the JVM and you must have a Java Development Kit (JDK) installed to use it. This is a bonus for users familiar with the Java platform as you can use the standard Java APIs in your build logic, such as custom task types and plugins. It also makes it easy to run Gradle on different platforms.

4.1.3 Tomcat server

Apache Tomcat (sometimes simply "Tomcat") is an open-source implementation of the Java Servlet, JavaServer Pages, Java Expression Language and WebSocket technologies.[3] Tomcat provides a "pure Java" HTTP web server environment in which Java code can run.

Figure 4.3 : – Tomcat logo.

The Apache Tomcat® software is an open source implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies. The Java

Servlet, JavaServer Pages, Java Expression Language and Java WebSocket specifications are developed under the Java Community Process.

4.1.4 Oracle RDBMS

Oracle Database is the first database designed for enterprise grid computing, the most flexible and cost effective way to manage information and applications. Enterprise grid computing creates large pools of industry-standard, modular storage and servers. With this architecture, each new system can be rapidly provisioned from the pool of components. There is no need for peak workloads, because capacity can be easily added or reallocated from the resource pools as needed.

ORACLE®

DATABASE

Figure 4.4 : – Oracle DB logo.

Oracle's revolutionary cloud database is self-driving, self-securing, self-repairing, and designed to eliminate error-prone manual data management. Easily deploy new or move your existing OLTP and data warehouse to the cloud. The secured, intelligent, highly available database in the cloud enables you to get more value from your data to grow your business.

4.1.5 Jenkins

Jenkins is a free and open source automation server. It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery. It is a server-based system that runs in servlet containers such as Apache Tomcat.

Figure 4.5 : – Jenkins logo.

The Jenkins project was originally named Hudson, and was renamed after a dispute with Oracle, which had forked the project and claimed rights to the project name.

4.2 Delivery steps of the new product

To deliver a new product to our master project, we go through several steps it's become ready for merge and included to the next release :

4.2.1 Reviewing phase

The reviewing phase is the first and the most important step of the process, since it's gather the long discussion about the source code choices, and the best way of solving the problems during the development process, because eventually you're code may seem on the good path for you, but others got a critical opinions about it.

4.2.2 Testing phase

Once we finished developing the solution, we check that our tests passed smoothly as we suggested on the design page, eventually the testing phase is divided on two main parts :

Integration test

- To make sure the integrated modules works fine individually.

Regression test

- To confirm that a recent code change has not adversely affected the existing features.

4.2.3 Approval and merging phase

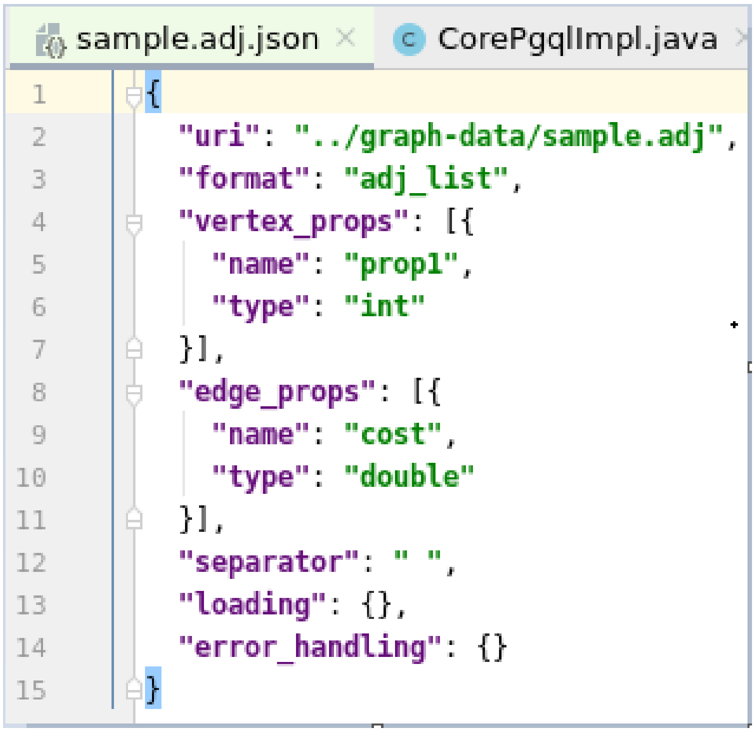
Finally when the tests passed successfully, the reviewers had a final examination to the code and approved the solution, once every member of the team approved, we update the change log of the new release and add the leader of the project to catch up the changes made and then I merge it to the master.

4.3 Graph presentation

During the testing and realisation phases, I was working locally on the Filesystem of the graphs that we had on the resources part of the project, the recognition of the graphs is divided on two files, one for the configuration and the other for parsing the data.

On the below explanation, I will illustrate the main form that I used during the demonstration process :

i. JSON configuration



```json
{
  "uri": "../graph-data/sample.adj",
  "format": "adj_list",
  "vertex_props": [{
    "name": "prop1",
    "type": "int"
  }],
  "edge_props": [{
    "name": "cost",
    "type": "double"
  }],
  "separator": " ",
  "loading": {},
  "error_handling": {}
}
```

Figure 4.6 : – JSON configuration.

ii. Data Parsing

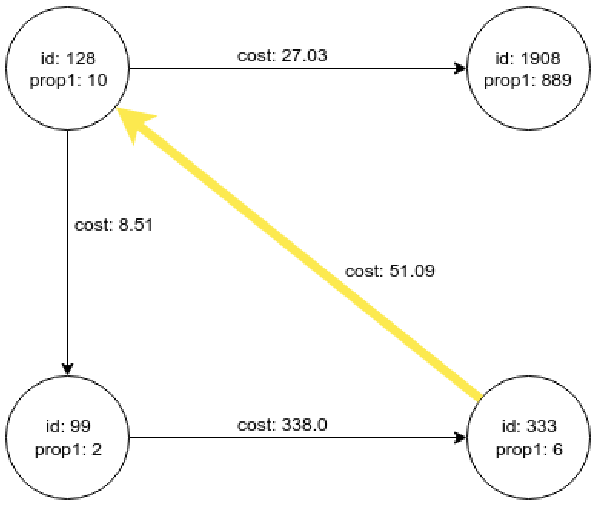
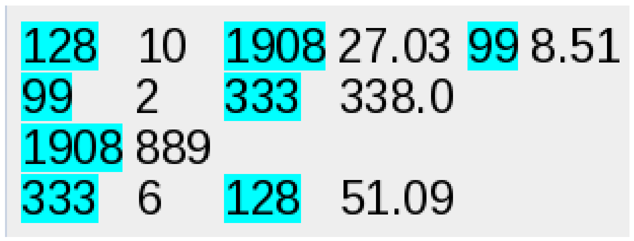


Figure 4.7 : – graph example.

For the above graph, the parsing data with the adjacency list form will be :



```
128   10   1908 27.03 99 8.51
99    2    333  338.0
1908 889
333   6    128  51.09
```

Figure 4.8 : – adjacency list data

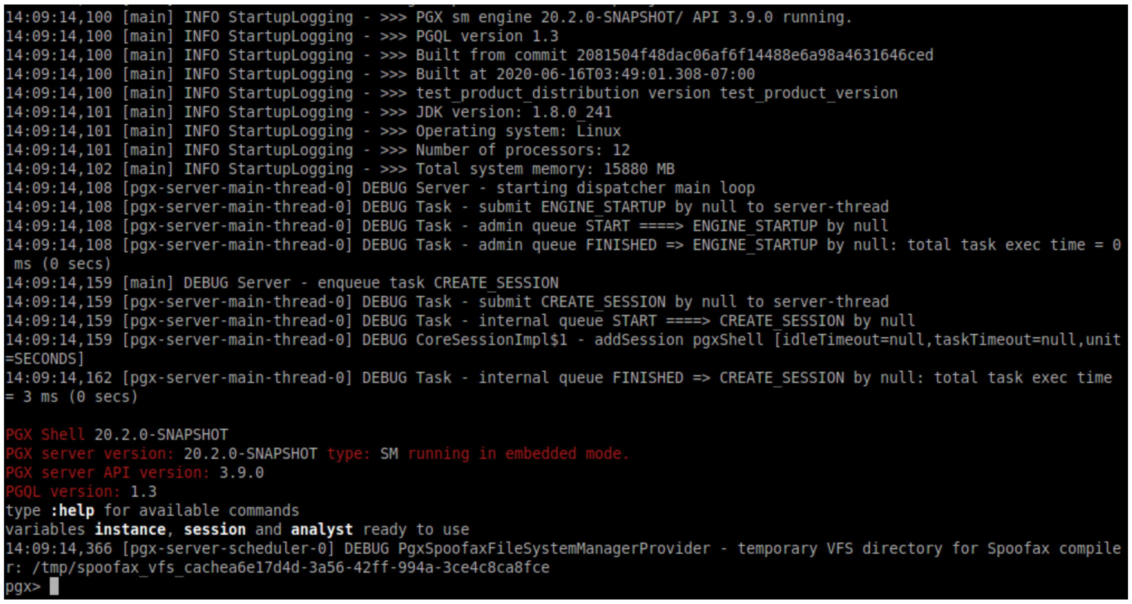
- The first column contains all the vertices of the graph with their properties.
- Each line contains the destination vertices of the edges with their properties if there is any.

On PGX project, we hold other forms to support graphs data like ; CSV files, binary files and two tables.

4.4 Demonstration

On this section, I will begin with a quick description of the experiment environment, the PGX groovy/java SHELL, then we illustrate a detailed demonstration on the GC basic use by the supporting admin APIs, and also a demonstration for the standard use of the GC service for PGX users .

4.4.1 PGX shell



```
14:09:14,100 [main] INFO StartupLogging - >>> PGX sm engine 20.2.0-SNAPSHOT/ API 3.9.0 running.
14:09:14,100 [main] INFO StartupLogging - >>> PGQL version 1.3
14:09:14,100 [main] INFO StartupLogging - >>> Built from commit 2081504f48dac06af6f14488e6a98a4631646ced
14:09:14,100 [main] INFO StartupLogging - >>> Built at 2020-06-16T03:49:01.308-07:00
14:09:14,100 [main] INFO StartupLogging - >>> test_product_distribution version test_product_version
14:09:14,101 [main] INFO StartupLogging - >>> JDK version: 1.8.0_241
14:09:14,101 [main] INFO StartupLogging - >>> Operating system: Linux
14:09:14,101 [main] INFO StartupLogging - >>> Number of processors: 12
14:09:14,102 [main] INFO StartupLogging - >>> Total system memory: 15880 MB
14:09:14,108 [pgx-server-main-thread-0] DEBUG Server - starting dispatcher main loop
14:09:14,108 [pgx-server-main-thread-0] DEBUG Task - submit ENGINE_STARTUP by null to server-thread
14:09:14,108 [pgx-server-main-thread-0] DEBUG Task - admin queue START ====> ENGINE_STARTUP by null
14:09:14,108 [pgx-server-main-thread-0] DEBUG Task - admin queue FINISHED => ENGINE_STARTUP by null: total task exec time = 0
 ms (0 secs)
14:09:14,159 [main] DEBUG Server - enqueue task CREATE_SESSION
14:09:14,159 [pgx-server-main-thread-0] DEBUG Task - submit CREATE_SESSION by null to server-thread
14:09:14,159 [pgx-server-main-thread-0] DEBUG Task - internal queue START ====> CREATE_SESSION by null
14:09:14,159 [pgx-server-main-thread-0] DEBUG CoreSessionImpl$1 - addSession pgxShell [idleTimeout=null,taskTimeout=null,unit
=SECONDS]
14:09:14,162 [pgx-server-main-thread-0] DEBUG Task - internal queue FINISHED => CREATE_SESSION by null: total task exec time
= 3 ms (0 secs)

PGX Shell 20.2.0-SNAPSHOT
PGX server version: 20.2.0-SNAPSHOT type: SM running in embedded mode.
PGX server API version: 3.9.0
PGQL version: 1.3
type :help for available commands
variables instance, session and analyst ready to use
14:09:14,366 [pgx-server-scheduler-0] DEBUG PgxSpoofaxFileSystemManagerProvider - temporary VFS directory for Spoofax compile
r: /tmp/spoofax_vfs_cachea6e17d4d-3a56-42ff-994a-3ce4c8ca8fce
pgx> 
```

Figure 4.9 : – PGX Shell Interface

The PGX shell is an interactive command-line application for interacting with the PGX runtime on local or remote computers. The PGX shell dynamically interprets command-line inputs from the user, executes them by invoking the underlying PGX functionality and can print results or process them further. The PGX shell provides a light-weight and interactive way of exercising PGX functionality without creating a Java application.

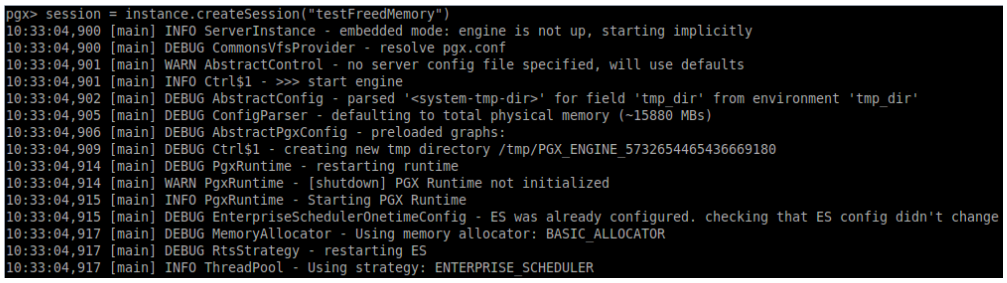
4.4.2 admin demonstration

For an admin use, we got access to the shared memory on the engine, so he can invoke the GC APIs to free the cached memory, on this experiment we explain how we can clear the memory on the server by this application on the graph we had on the resources file, we illustrate the taken steps by shell screens and memory states :

4.4.2.1 Loading phase

As first step, we begin by creating a new session as the workflow for the user, then we can load some graphs and run any features we want, but on our part we will focus on the memory allocations, and how it's being cleared by those Admin APIs.

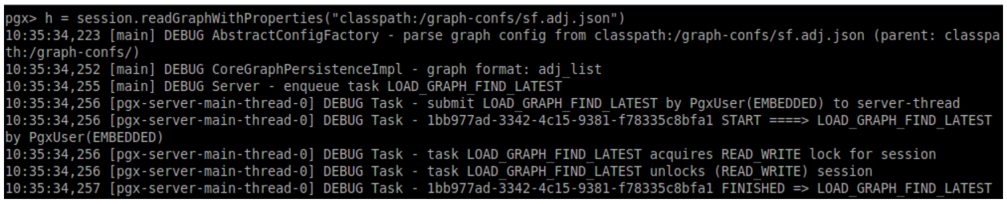
- **First step** create session ≪ testMemoryFreed ≫.



```
pgx> session = instance.createSession("testFreedMemory")
10:33:04,900 [main] INFO ServerInstance - embedded mode: engine is not up, starting implicitly
10:33:04,900 [main] DEBUG CommonsVfsProvider - resolve pgx.conf
10:33:04,901 [main] WARN AbstractControl - no server config file specified, will use defaults
10:33:04,901 [main] INFO Ctrl$1 - >>> start engine
10:33:04,902 [main] DEBUG AbstractConfig - parsed '<system-tmp-dir>' for field 'tmp_dir' from environment 'tmp_dir'
10:33:04,905 [main] DEBUG ConfigParser - defaulting to total physical memory (~15880 MBs)
10:33:04,906 [main] DEBUG AbstractPgxConfig - preloaded graphs:
10:33:04,909 [main] DEBUG Ctrl$1 - creating new tmp directory /tmp/PGX_ENGINE_5732654465436669180
10:33:04,914 [main] DEBUG PgxRuntime - restarting runtime
10:33:04,914 [main] WARN PgxRuntime - [shutdown] PGX Runtime not initialized
10:33:04,915 [main] INFO PgxRuntime - Starting PGX Runtime
10:33:04,915 [main] DEBUG EnterpriseSchedulerOnetimeConfig - ES was already configured. checking that ES config didn't change
10:33:04,917 [main] DEBUG MemoryAllocator - Using memory allocator: BASIC_ALLOCATOR
10:33:04,917 [main] DEBUG RtsStrategy - restarting ES
10:33:04,917 [main] INFO ThreadPool - Using strategy: ENTERPRISE_SCHEDULER
```

Figure 4.10 : – Session Created

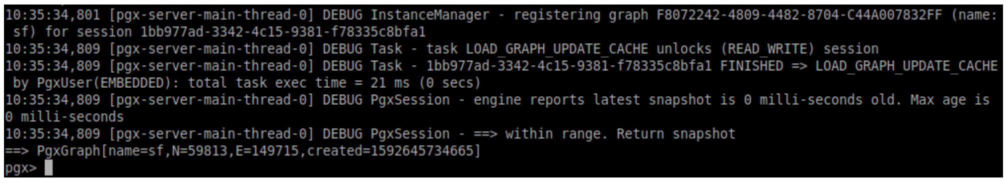
- **Second step** load an heterogenuous graph.



```
pgx> h = session.readGraphWithProperties("classpath:/graph-confs/sf.adj.json")
10:35:34,223 [main] DEBUG AbstractConfigFactory - parse graph config from classpath:/graph-confs/sf.adj.json (parent: classpa
th:/graph-confs/)
10:35:34,252 [main] DEBUG CoreGraphPersistenceImpl - graph format: adj_list
10:35:34,255 [main] DEBUG Server - enqueue task LOAD_GRAPH_FIND_LATEST
10:35:34,256 [pgx-server-main-thread-0] DEBUG Task - submit LOAD_GRAPH_FIND_LATEST by PgxUser(EMBEDDED) to server-thread
10:35:34,256 [pgx-server-main-thread-0] DEBUG Task - 1bb977ad-3342-4c15-9381-f78335c8bfa1 START ====> LOAD_GRAPH_FIND_LATEST
by PgxUser(EMBEDDED)
10:35:34,256 [pgx-server-main-thread-0] DEBUG Task - task LOAD_GRAPH_FIND_LATEST acquires READ_WRITE lock for session
10:35:34,256 [pgx-server-main-thread-0] DEBUG Task - task LOAD_GRAPH_FIND_LATEST unlocks (READ_WRITE) session
10:35:34,257 [pgx-server-main-thread-0] DEBUG Task - 1bb977ad-3342-4c15-9381-f78335c8bfa1 FINISHED => LOAD_GRAPH_FIND_LATEST
```

Figure 4.11 : – Graph Loaded

The screen above show how we can load our graph by the related classpath, the loaded graph has the given properties:



```
10:35:34,801 [pgx-server-main-thread-0] DEBUG InstanceManager - registering graph F8072242-4809-4482-8704-C44A007832FF (name:
 sf) for session 1bb977ad-3342-4c15-9381-f78335c8bfa1
10:35:34,809 [pgx-server-main-thread-0] DEBUG Task - task LOAD_GRAPH_UPDATE_CACHE unlocks (READ_WRITE) session
10:35:34,809 [pgx-server-main-thread-0] DEBUG Task - 1bb977ad-3342-4c15-9381-f78335c8bfa1 FINISHED => LOAD_GRAPH_UPDATE_CACHE
 by PgxUser(EMBEDDED): total task exec time = 21 ms (0 secs)
10:35:34,809 [pgx-server-main-thread-0] DEBUG PgxSession - engine reports latest snapshot is 0 milli-seconds old. Max age is
0 milli-seconds
10:35:34,809 [pgx-server-main-thread-0] DEBUG PgxSession - ==> within range. Return snapshot
==> PgxGraph[name=sf,N=59813,E=149715,created=1592645734665]
pgx>
```

Figure 4.12 : – Graph properties

The graph above that had a huge number of vertices and edges consumes 5 mb on the memory.

After the loading process the memory state becomes as the diagram below shows:

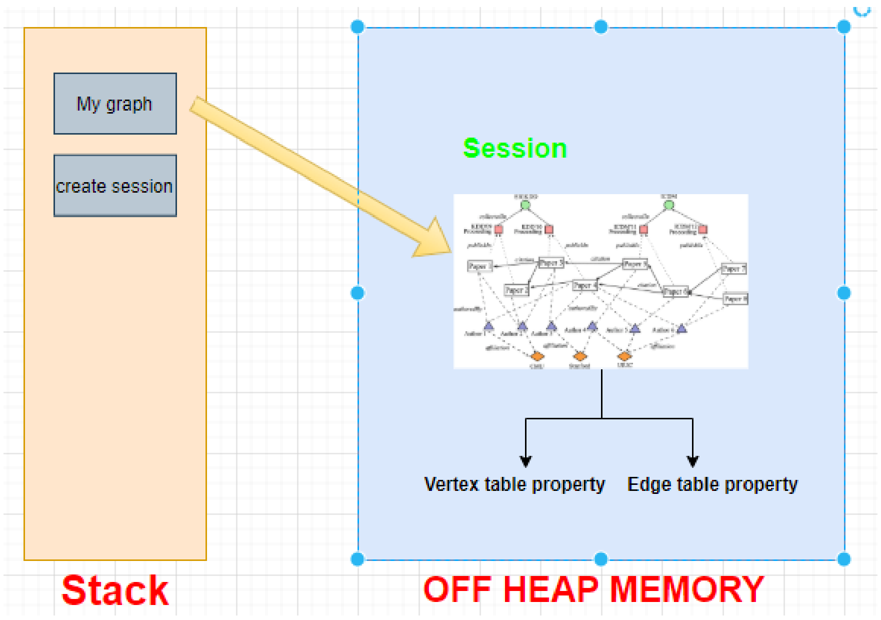
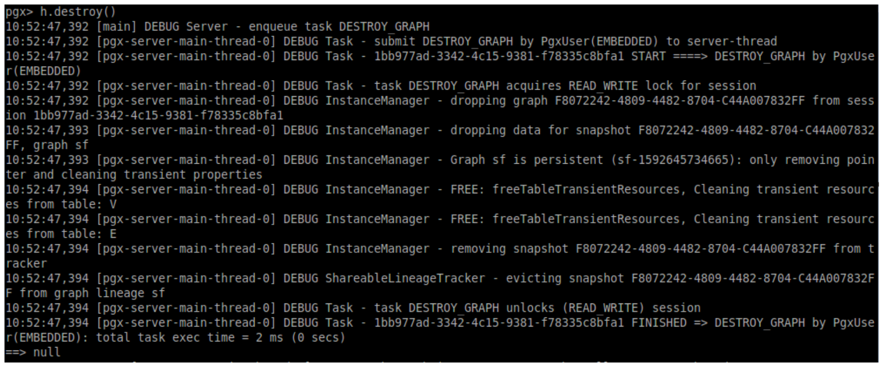


Figure 4.13 : – Memory state after loading the graph

4.4.2.2 Destroying phase

Once we load our graph, we can run any PGQL queries or algorithms on the graphs, then after we finished the process, the graph destroyed automatically once we leave the session or we can destroy it manually by the destroy() method as the screen below showed :



```
pgx> h.destroy()
10:52:47,392 [main] DEBUG Server - enqueue task DESTROY_GRAPH
10:52:47,392 [pgx-server-main-thread-0] DEBUG Task - submit DESTROY_GRAPH by PgxUser(EMBEDDED) to server-thread
10:52:47,392 [pgx-server-main-thread-0] DEBUG Task - 1bb977ad-3342-4c15-9381-f78335c8bfa1 START ====> DESTROY_GRAPH by PgxUse
r(EMBEDDED)
10:52:47,392 [pgx-server-main-thread-0] DEBUG Task - task DESTROY_GRAPH acquires READ_WRITE lock for session
10:52:47,392 [pgx-server-main-thread-0] DEBUG InstanceManager - dropping graph F8072242-4809-4482-8704-C44A007832FF from sess
ion 1bb977ad-3342-4c15-9381-f78335c8bfa1
10:52:47,393 [pgx-server-main-thread-0] DEBUG InstanceManager - dropping data for snapshot F8072242-4809-4482-8704-C44A007832
FF, graph sf
10:52:47,393 [pgx-server-main-thread-0] DEBUG InstanceManager - Graph sf is persistent (sf-1592645734665): only removing poin
ter and cleaning transient properties
10:52:47,394 [pgx-server-main-thread-0] DEBUG InstanceManager - FREE: freeTableTransientResources, Cleaning transient resourc
es from table: V
10:52:47,394 [pgx-server-main-thread-0] DEBUG InstanceManager - FREE: freeTableTransientResources, Cleaning transient resourc
es from table: E
10:52:47,394 [pgx-server-main-thread-0] DEBUG InstanceManager - removing snapshot F8072242-4809-4482-8704-C44A007832FF from t
racker
10:52:47,394 [pgx-server-main-thread-0] DEBUG ShareableLineageTracker - evicting snapshot F8072242-4809-4482-8704-C44A007832F
F from graph lineage sf
10:52:47,394 [pgx-server-main-thread-0] DEBUG Task - task DESTROY_GRAPH unlocks (READ_WRITE) session
10:52:47,394 [pgx-server-main-thread-0] DEBUG Task - 1bb977ad-3342-4c15-9381-f78335c8bfa1 FINISHED => DESTROY_GRAPH by PgxUse
r(EMBEDDED): total task exec time = 2 ms (0 secs)
==> null
```

Figure 4.14 : – graph destroyed

After we destroy our graph, we technically erase the reference but the graph physically stay holding the memory as the memory state shows on the figure below:

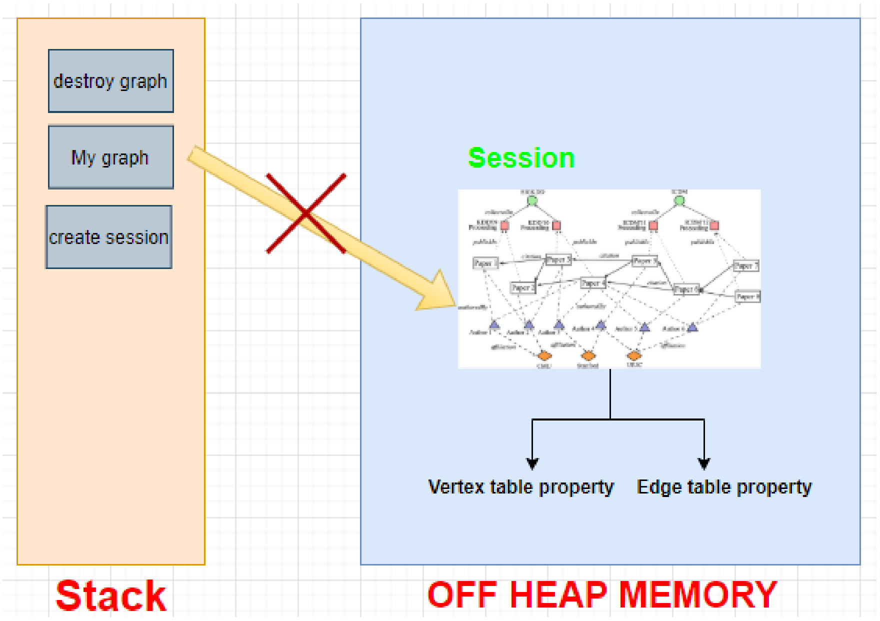
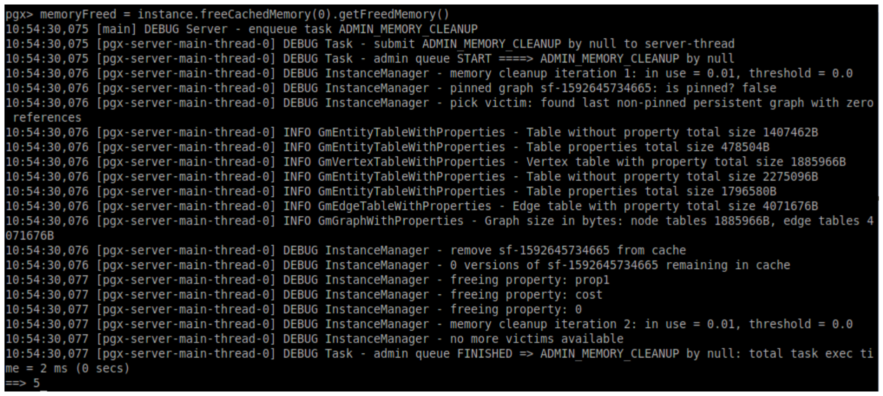


Figure 4.15 : – Memory State after destroying the graph

4.4.2.3 freeing memory phase

As a last step of the demonstration, we' re going to see how the admin API free the memory from the server, by the memory cleanup process as the figure below shows :



```
pgx> memoryFreed = instance.freeCachedMemory(0).getFreedMemory()
10:54:30,075 [main] DEBUG Server - enqueue task ADMIN_MEMORY_CLEANUP
10:54:30,075 [pgx-server-main-thread-0] DEBUG Task - submit ADMIN_MEMORY_CLEANUP by null to server-thread
10:54:30,075 [pgx-server-main-thread-0] DEBUG Task - admin queue START ====> ADMIN_MEMORY_CLEANUP by null
10:54:30,076 [pgx-server-main-thread-0] DEBUG InstanceManager - memory cleanup iteration 1: in use = 0.01, threshold = 0.0
10:54:30,076 [pgx-server-main-thread-0] DEBUG InstanceManager - pinned graph sf-1592645734665: is pinned? false
10:54:30,076 [pgx-server-main-thread-0] DEBUG InstanceManager - pick victim: found last non-pinned persistent graph with zero
 references
10:54:30,076 [pgx-server-main-thread-0] INFO GmEntityTableWithProperties - Table without property total size 1407462B
10:54:30,076 [pgx-server-main-thread-0] INFO GmEntityTableWithProperties - Table properties total size 478504B
10:54:30,076 [pgx-server-main-thread-0] INFO GmVertexTableWithProperties - Vertex table with property total size 1885966B
10:54:30,076 [pgx-server-main-thread-0] INFO GmEntityTableWithProperties - Table without property total size 2275096B
10:54:30,076 [pgx-server-main-thread-0] INFO GmEntityTableWithProperties - Table properties total size 1796580B
10:54:30,076 [pgx-server-main-thread-0] INFO GmEdgeTableWithProperties - Edge table with property total size 4071676B
10:54:30,076 [pgx-server-main-thread-0] INFO GmGraphWithProperties - Graph size in bytes: node tables 1885966B, edge tables 4
071676B
10:54:30,076 [pgx-server-main-thread-0] DEBUG InstanceManager - remove sf-1592645734665 from cache
10:54:30,076 [pgx-server-main-thread-0] DEBUG InstanceManager - 0 versions of sf-1592645734665 remaining in cache
10:54:30,077 [pgx-server-main-thread-0] DEBUG InstanceManager - freeing property: prop1
10:54:30,077 [pgx-server-main-thread-0] DEBUG InstanceManager - freeing property: cost
10:54:30,077 [pgx-server-main-thread-0] DEBUG InstanceManager - freeing property: 0
10:54:30,077 [pgx-server-main-thread-0] DEBUG InstanceManager - memory cleanup iteration 2: in use = 0.01, threshold = 0.0
10:54:30,077 [pgx-server-main-thread-0] DEBUG InstanceManager - no more victims available
10:54:30,077 [pgx-server-main-thread-0] DEBUG Task - admin queue FINISHED => ADMIN_MEMORY_CLEANUP by null: total task exec ti
me = 2 ms (0 secs)
==> 5
```

Figure 4.16 : – The free memory process

- **Technical explanation :**

we need to call the GC API with the overide threshold ≪ freeCachedMemory(0) ≫ to make sure that we're going to clean everything on the server, and for the Memory Cleanup logic we can divide the explanation process on three phases :

1. **iterating :**
   As first step, we begin by iterating through the graphs been loaded, on this case we found our heterogeneous graph, and since it's non referenced one, it will be picked as victim.

2. **memory accumulating :**
   Once we picked the victim, we get the size of all the tables that hold the memory on the graph object(vertex table property, edge table property), so we can accumulated as a victim size and hold it on our Cached Statistics result.

3. **free and check :**
   On the last iteration, we begin by freeing the memory of the detected victim, then we check on the off-heap memory to see if there's any other victims to catch, on our case no more victims available since we load just one graph.

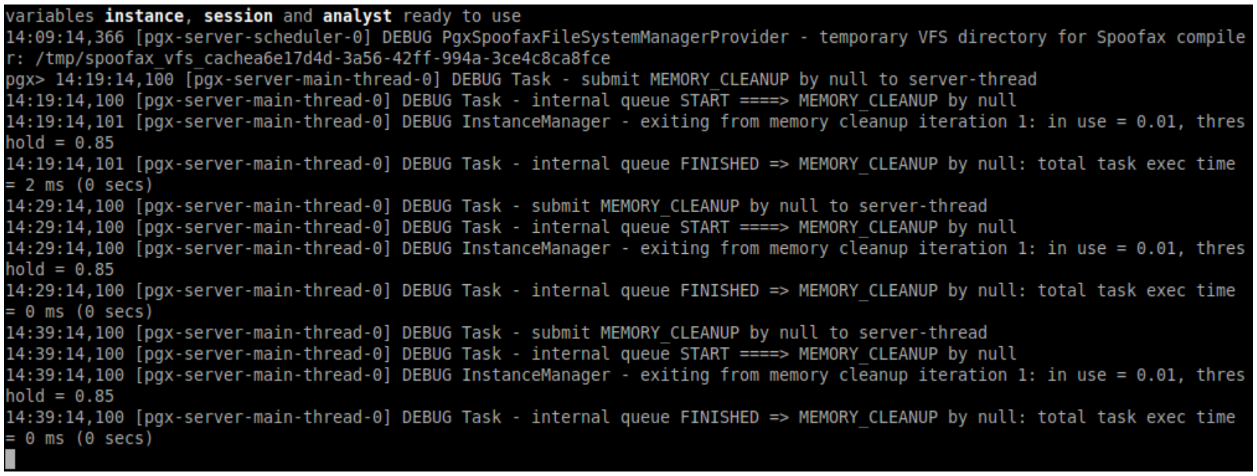
4.4.3 User demonstration

For the user experience, we set the memoryCleanup process to be invoked by those two parameters :

• **memoryCleanup interval :** It's the time factor that specify the cycles duration for invoking the thread process.

• **release threshold :** It's the factor that decide either we begin picking the graphs from the cached memory and free them or not .

the figure below will illustrate the display information on the groovy SHELL :



```
variables instance, session and analyst ready to use
14:09:14,366 [pgx-server-scheduler-0] DEBUG PgxSpoofaxFileSystemManagerProvider - temporary VFS directory for Spoofax compile
r: /tmp/spoofax_vfs_cachea6e17d4d-3a56-42ff-994a-3ce4c8ca8fce
pgx> 14:19:14,100 [pgx-server-main-thread-0] DEBUG Task - submit MEMORY_CLEANUP by null to server-thread
14:19:14,100 [pgx-server-main-thread-0] DEBUG Task - internal queue START ====> MEMORY_CLEANUP by null
14:19:14,101 [pgx-server-main-thread-0] DEBUG InstanceManager - exiting from memory cleanup iteration 1: in use = 0.01, thres
hold = 0.85
14:19:14,101 [pgx-server-main-thread-0] DEBUG Task - internal queue FINISHED => MEMORY_CLEANUP by null: total task exec time
= 2 ms (0 secs)
14:29:14,100 [pgx-server-main-thread-0] DEBUG Task - submit MEMORY_CLEANUP by null to server-thread
14:29:14,100 [pgx-server-main-thread-0] DEBUG Task - internal queue START ====> MEMORY_CLEANUP by null
14:29:14,100 [pgx-server-main-thread-0] DEBUG InstanceManager - exiting from memory cleanup iteration 1: in use = 0.01, thres
hold = 0.85
14:29:14,100 [pgx-server-main-thread-0] DEBUG Task - internal queue FINISHED => MEMORY_CLEANUP by null: total task exec time
= 0 ms (0 secs)
14:39:14,100 [pgx-server-main-thread-0] DEBUG Task - submit MEMORY_CLEANUP by null to server-thread
14:39:14,100 [pgx-server-main-thread-0] DEBUG Task - internal queue START ====> MEMORY_CLEANUP by null
14:39:14,100 [pgx-server-main-thread-0] DEBUG InstanceManager - exiting from memory cleanup iteration 1: in use = 0.01, thres
hold = 0.85
14:39:14,100 [pgx-server-main-thread-0] DEBUG Task - internal queue FINISHED => MEMORY_CLEANUP by null: total task exec time
= 0 ms (0 secs)
```

Figure 4.17 : – Memory Cleanup cycles

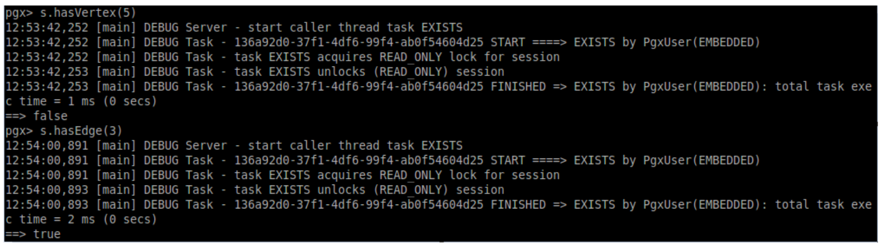
For the standard use case of the GC service on the engine, the team leader decide to set the memory cleanup process each 10 min on the engine, and configured with release threshold of 0.85 / 1 to free the cached memory on the server.

on the above experiment, no memory got freed since there's no loaded graphs on the server, but the same basic process on the admin demonstration for freeing the memory it's applied, with the used of those standard configurations.

4.5 Other Unrelated tasks

4.5.1 Support hasVertex(id) and hasEdge(id) for partitioned graphs

Since we support the new model of graphs (heterogeneous one), we create several tasks to support the features related to that type of graphs, and the one that I support is to check if the vertex and edge with given Id exist or not and return the boolean statement as result.



```
pgx> s.hasVertex(5)
12:53:42,252 [main] DEBUG Server - start caller thread task EXISTS
12:53:42,252 [main] DEBUG Task - 136a92d0-37f1-4df6-99f4-ab0f54604d25 START ====> EXISTS by PgxUser(EMBEDDED)
12:53:42,252 [main] DEBUG Task - task EXISTS acquires READ_ONLY lock for session
12:53:42,253 [main] DEBUG Task - task EXISTS unlocks (READ_ONLY) session
12:53:42,253 [main] DEBUG Task - 136a92d0-37f1-4df6-99f4-ab0f54604d25 FINISHED => EXISTS by PgxUser(EMBEDDED): total task exe
c time = 1 ms (0 secs)
==> false
pgx> s.hasEdge(3)
12:54:00,891 [main] DEBUG Server - start caller thread task EXISTS
12:54:00,891 [main] DEBUG Task - 136a92d0-37f1-4df6-99f4-ab0f54604d25 START ====> EXISTS by PgxUser(EMBEDDED)
12:54:00,891 [main] DEBUG Task - task EXISTS acquires READ_ONLY lock for session
12:54:00,893 [main] DEBUG Task - task EXISTS unlocks (READ_ONLY) session
12:54:00,893 [main] DEBUG Task - 136a92d0-37f1-4df6-99f4-ab0f54604d25 FINISHED => EXISTS by PgxUser(EMBEDDED): total task exe
c time = 2 ms (0 secs)
==> true
```

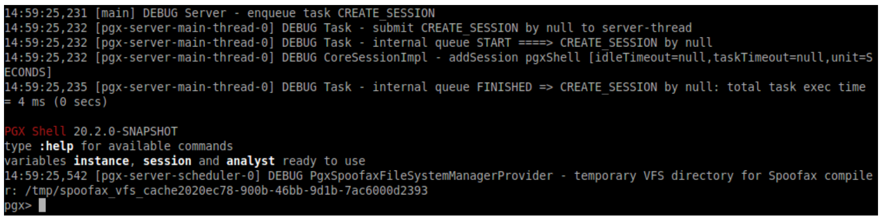
Figure 4.18 : – hasVertex(id) and hasEdge(id) output

4.5.2 Fixing bugs

Each time we found a new bug, we need to fix it as quickly as possible, for not falling into more complicated one, during the process I worked on several tasks related to bugs fixing, such as the regression failed test that caused by java 8 precision on the nano seconds part, and the solution was to drop that part from the time output to become handled by the test.

4.5.3 Configure shell output

This specific task was requested from the product team for making our shell output versions message configurable, those too many Shell versions made the clients confused about the usage process, so as solution I configure the display message on our engine, and every possible way of the output can be decided by the product team, as the example below show the displayed message after the configuration :



```
14:59:25,231 [main] DEBUG Server - enqueue task CREATE_SESSION
14:59:25,232 [pgx-server-main-thread-0] DEBUG Task - submit CREATE_SESSION by null to server-thread
14:59:25,232 [pgx-server-main-thread-0] DEBUG Task - internal queue START ====> CREATE_SESSION by null
14:59:25,232 [pgx-server-main-thread-0] DEBUG CoreSessionImpl - addSession pgxShell [idleTimeout=null,taskTimeout=null,unit=S
ECONDS]
14:59:25,235 [pgx-server-main-thread-0] DEBUG Task - internal queue FINISHED => CREATE_SESSION by null: total task exec time
= 4 ms (0 secs)

PGX Shell 20.2.0-SNAPSHOT
type :help for available commands
variables instance, session and analyst ready to use
14:59:25,542 [pgx-server-scheduler-0] DEBUG PgxSpoofaxFileSystemManagerProvider - temporary VFS directory for Spoofax compile
r: /tmp/spoofax_vfs_cache2020ec78-900b-46bb-9d1b-7ac6000d2393
pgx>
```

Figure 4.19 : – configured shell output

Conclusion

Through this chapter, we were able to set up the context of the prototype and the several steps for achieving the final product, then we hold a practical demonstrations which were the subject for the improvements we have made to the system. Each case application, deals with one aspect of the modifications made. The goal is to demonstrate the interest of our end-of-studies internship through examples.

These application cases also provided the opportunity to evaluate the prototype in the face of existing solutions. Our prototype stood out from the Most of them. and also to show the interest of the project and its potential in the future.

General conclusion

This thesis reports the progress of our graduation project carried out at Oracle. The objective of the project was to improve the resource manager prototype on Oracle's PGX project, in order to enforce the resources limits on the shared memory engine, the support of the GC APIs to manage the off heap memory and increase the performance for the concurrent usage of the server for several clients.

The purpose of this report was to describe the tasks I carried out during my internship at Oracle Labs Morocco, but the version project should not stop there. Indeed the next step will be to perform the integration tests, before submitting it to the regression test set, and then finally put it into production and ensure the warranty period on delivery.

I had the pleasure of working on the Resource manager project with the shared memory team composite of five people, myself included, who did their best to put me on the right track and familiarize myself with the project, and to whom I am very grateful.

To conclude, By doing my end-of-studies internship to obtain my engineering diploma, in a company whose reputation and scope are no longer to be described as Oracle, allowed me not only to acquire a strong experience in the technical side, but also to enjoy a great professional maturity. Also, I would like to express my complete satisfaction with my internship at Oracle Labs Morocco, and I hope that this experience is the beginning of a future collaboration, which will allow on the one hand to refine my theoretical knowledge, and on the other hand to continue on my momentum, and continue the work that I initiated on this version project.

Bibliography and webography

Bibliography

Books

- [B.1] : Clean Code *[2009 By Robert C. Martin]*
- [B.2] : The Garbage Collection Handbook *[2012 by Richard Jones, Antony Hosking, and Eliot Moss]*
- [B.3] : Java: The Complete Reference, Eleventh Edition, 11th Edition *[2019 by McGraw-Hill]*

Webography

- [W.1] : Oracle corporation,
  `https://en.wikipedia.org/wiki/Oracle_Corporation` *(Last visit 5 mars)*
- [W.2] : Oracle acquisitions,
  `https://en.wikipedia.org/wiki/Oracle_Corporation,` *(Last visit 8 mars)*,
- [W.3] : Oracle Labs : presentation and strategy,
  `https://labs.oracle.com/pls/apex/f?p=LABS:10:0:::::` *(Last visit 15 mars)*
- [W.4] : Oracle Labs : projects,
  `https://labs.oracle.com/pls/apex/f?p=94065:30:::NO:::` *(Last visit 15 mars)*
- [W.5] : Méthodologie Kanban,
  `https://fr.wikipedia.org/wiki/Kanban`$_($*dC*3*A*9*veloppement*$)$ ***(Last visit 1 mars)***
- [W.6] : Jira Software,
  `https://fr.atlassian.com/software/jira` *(Last visit 1 mars)*
- [W.7] : Confluence Software,
  `https://fr.atlassian.com/software/confluence` *(Last visit 1 mars)*

- [W.8] : Bitbucket Software,
  https://fr.atlassian.com/software/bitbucket *(Last visit 4 mars)*
- [W.9] : Jira Software,
  https://fr.atlassian.com/software/jira *(Last visit 4 mars)*
- [W.10] : PGX overview,
  https://www.oracle.com/technetwork/oracle-labs/parallel-graph-analytix/overview/,
  (Last visit 15 mai)
- [W.11] : Git,
  https://fr.wikipedia.org/wiki/Git, *(Last visit 20 mai)*
- [W.10] : Visual paradign online site,
  https://online.visual-paradigm.com/ *(Last visit 20 mai)*

  :

Appendices

Annex 1 : on-heap and off-heap memory

On-heap memory

Each newly created class instance is located in the eden space of young generation. If it's still in used in the moment of Garbage Collection, it's promoted to the next space. Finally, if it's a long lived object, it's moved to the old generation where the GC are less frequent and more susceptible to produce longer stop-the-world events (old generation is usually much bigger than each of previous spaces). If you want to learn more about it, you can read the post about Generations in JVM.

After this short reminder we could characterize on-heap memory as a memory present in the JVM and managed automatically by the Garbage Collector.

Off-heap memory

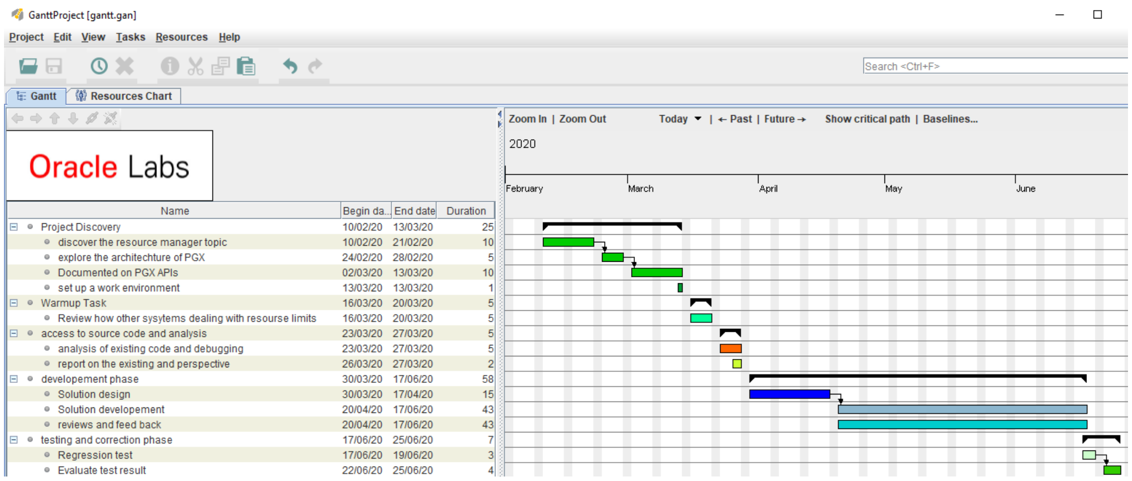
At first glance everything seems to be fine - objects are located and deallocated automatically, on-heap is freely configurable through appropriated options. But sometimes it's not enough, especially when we need to: cache a lot of data without increasing GC pauses, share cached data between JVMs or add a persistence layer in memory resistant to JVM crashes. In all mentioned cases off-heap memory is one of possible solutions.

As you can imagine, the off-heap memory stores the data outside the heap in OS memory part. Because there are no the JVM, the data must be stored in specific format that is an array of bytes. So using the off-heap memory in JVM languages programs introduces the overhead of serializing/deserializing these arrays to corresponding objects every time with additional cost of going outside the JVM and dealing with native memory. And because this space is out of JVM it can follow its own rules and bring other problems to programmers as Big-Endian and Little-Endian one.

However the use off-heap can help to reduce GC pauses (especially in large heaps). It also allows different process to share the data stored in memory (e.g. C++ program and Scala one). In additional, off-heap memory helps the data to survive JVM crashes. With that it's possible to have a long living hot cache.

Annex 2 : tools and used technologies

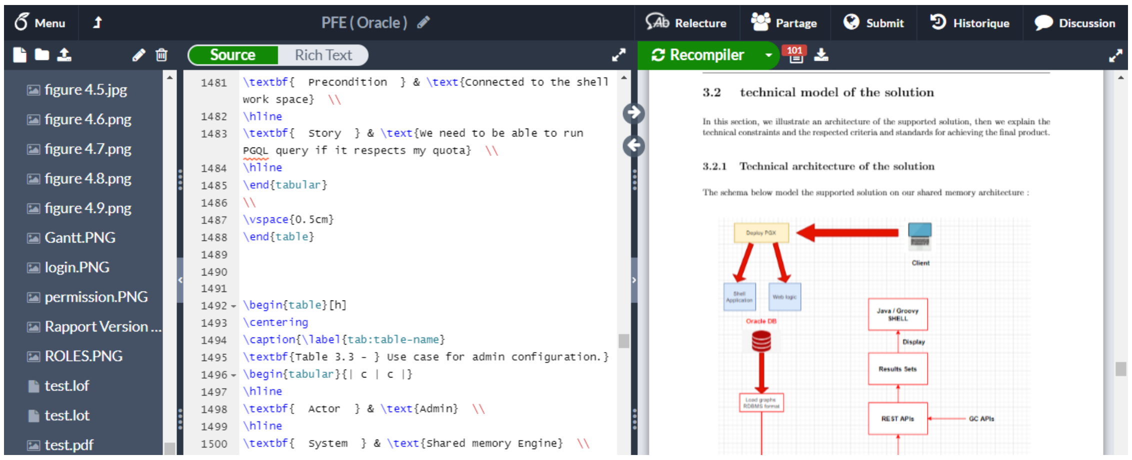
GanttProject



Name	Begin da...	End date	Duration
Project Discovery	10/02/20	13/03/20	25
discover the resource manager topic	10/02/20	21/02/20	10
explore the architechture of PGX	24/02/20	28/02/20	5
Documented on PGX APIs	02/03/20	13/03/20	10
set up a work environment	13/03/20	13/03/20	1
Warmup Task	16/03/20	20/03/20	5
Review how other sysytems dealing with resourse limits	16/03/20	20/03/20	5
access to source code and analysis	23/03/20	27/03/20	5
analysis of existing code and debugging	23/03/20	27/03/20	5
report on the existing and perspective	26/03/20	27/03/20	2
developement phase	30/03/20	17/06/20	58
Solution design	30/03/20	17/04/20	15
Solution developement	20/04/20	17/06/20	43
reviews and feed back	20/04/20	17/06/20	43
testing and correction phase	17/06/20	25/06/20	7
Regression test	17/06/20	19/06/20	3
Evaluate test result	22/06/20	25/06/20	4

Visual-paradigm

LATEX



```
\textbf{  Precondition  } & \text{Connected to the shell
work space}  \\
\hline
\textbf{  Story  } & \text{We need to be able to run
PGQL query if it respects my quota}  \\
\hline
\end{tabular}
\\
\vspace{0.5cm}
\end{table}



\begin{table}[h]
\centering
\caption{\label{tab:table-name}
\textbf{Table 3.3 - } Use case for admin configuration.}
\begin{tabular}{| c | c |}
\hline
\textbf{  Actor  } & \text{Admin}  \\
\hline
\textbf{  System  } & \text{Shared memory Engine}  \\
```

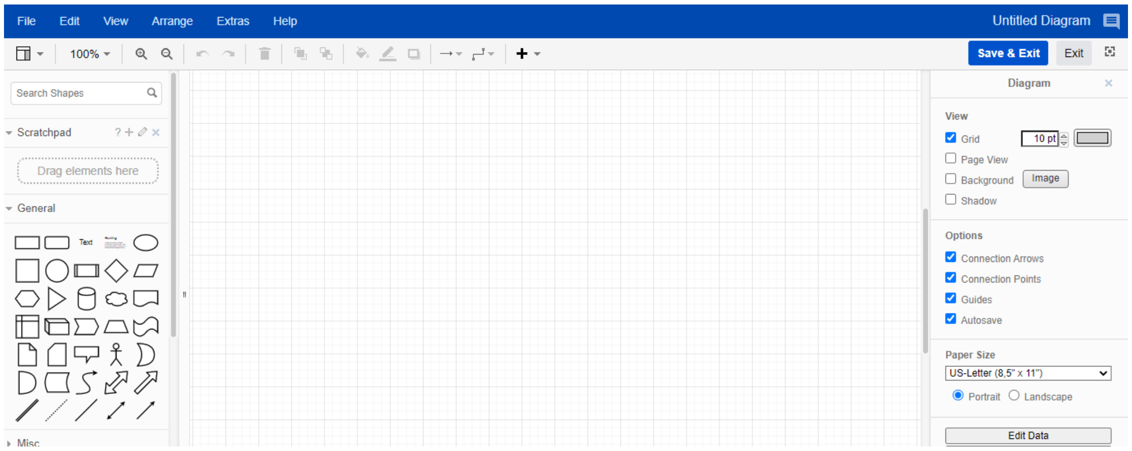
3.2 technical model of the solution

In this section, we illustrate an architecture of the supported solution, then we explain the technical constraints and the respected criteria and standards for achieving the final product.

3.2.1 Technical architecture of the solution

The schema below model the supported solution on our shared memory architecture :

Draw.io



View
[✓] Grid 10 pt
[ ] Page View
[ ] Background Image
[ ] Shadow

Options
[✓] Connection Arrows
[✓] Connection Points
[✓] Guides
[✓] Autosave

Paper Size
US-Letter (8,5" x 11")
[✓ Portrait] Landscape