REG : P15/1667/2019
NAME : Wagura James Kiarie

Project files : here

Lab X is a software development company. They have a distributed database for their project management software. It stores employee details, project details and Assignation details.
They have branches across different locations ; Nairobi, Nakuru and Kericho. Their headoffice is located at the Nairobi Branch. Within that project management app, the following 2 queries get executed :

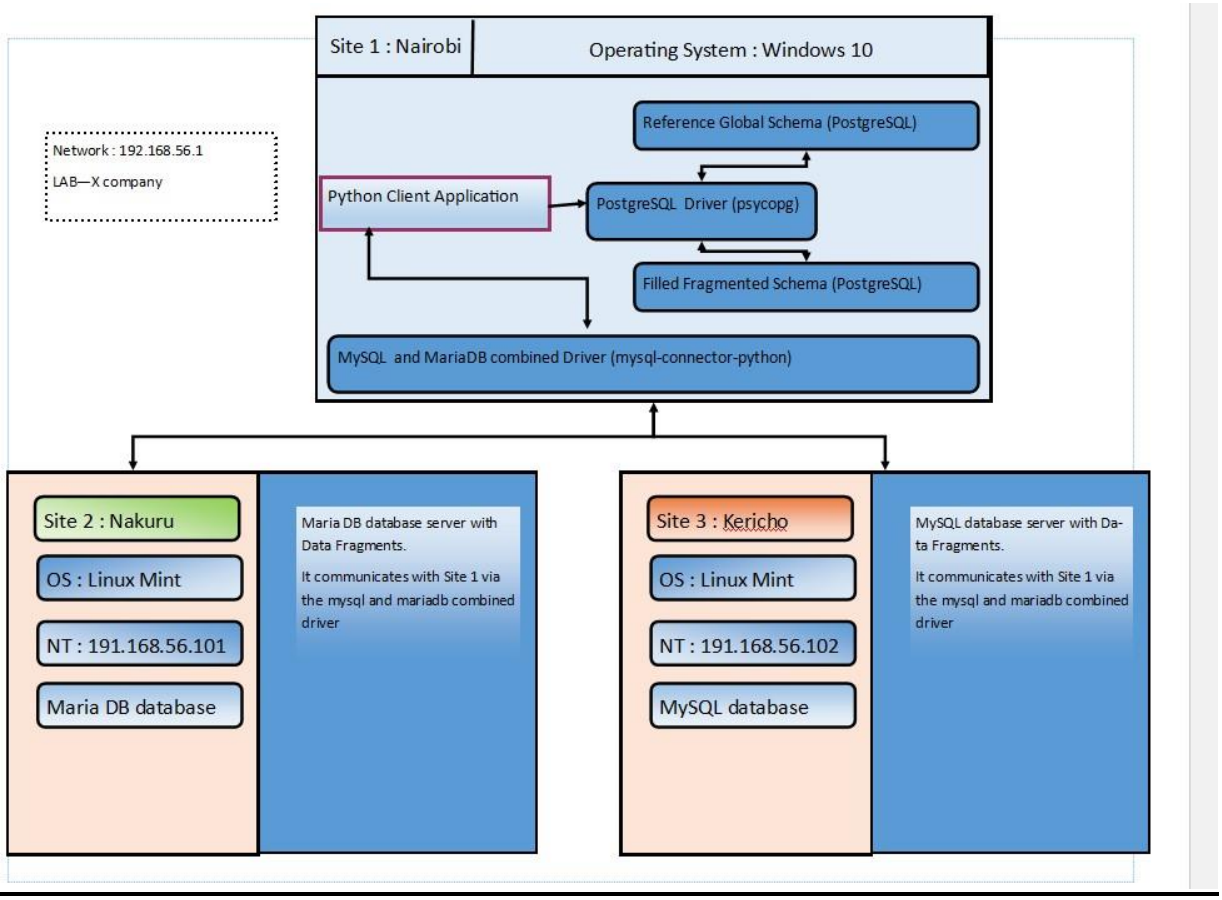
- App 1 : The head office should be able to list all the projects that have a budget of at least 200,000/=. Regardless of the project location.

- App 2 : Branch managers should be able to list all the projects being done in their location

Here is the access frequency of the above queries:
Where
- R1 = projects table
- R2 = job descriptions table
- R3 = employees table
- R4 = assignations table

	Relations Affected				Sites		
	R1	R2	R3	R4	Nairobi	Nakuru	Kericho
App_1	1	0	0	0	20	0	0
App_2	1	0	0	0	15	16	17

Overall Design Architecture of Distributed DB

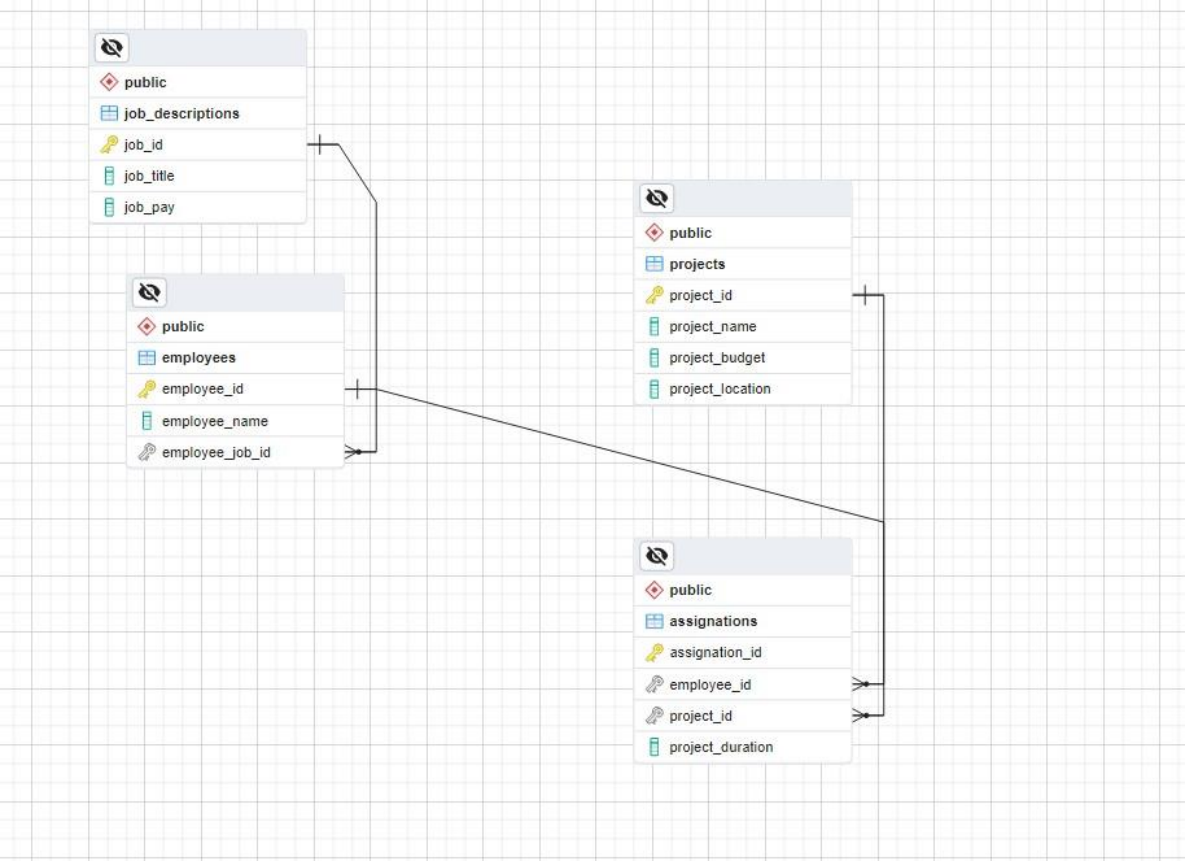


The Distributed database was comprised of 3 sites :

- **Site 1 :** Runs in Nairobi, has Windows 10 operating system and uses PostgreSql database platform
- **Site 2 :** Runs in Nakuru, has Linux Mint Operating System and uses MariaDB database
- **Site 3 :** Runs in Kericho, has Linux Mint Operating System and uses MySQL database

Database Schema

You can access the global schema here.



The fragmentation process :

- Re-writing App_1 as a global SQL query:
  SELECT *
  FROM projects
  WHERE projects.project_budget >= 200000

- Re-writing App_2 as a global SQL query:
  SELECT *
  FROM projects
  WHERE projects.project_location = 'Nakuru'

- Extracting simple predicates from App_1 and App_2 storing them in an empty set called SP
  SP = { p1, p2, p3, p4, p5 } where
  - P1 : projects.project_location = 'Nairobi'
  - P2 : projects.project_location = 'Nakuru'
  - P3 : projects.project_location = 'Kericho'
  - P4 : projects.project_budget >= 200000
  - P5 : projects.project_budget < 200000

- Deriving relevant minterm predicates from the list of simple predicates SP
    - M1 : P1 AND P4
    - M2 : P1 AND P5
    - M3 : P2 AND P4
    - M4 : P2 AND P5
    - M5 : P3 AND P4
    - M6 : P3 AND P5

- Creating the fragments according to the minterm fragments
    - Fragment 1 : SELECT * FROM projects WHERE M1
    - Fragment 2 : SELECT * FROM projects WHERE M2
    - Fragment 3 : SELECT * FROM projects WHERE M3
    - Fragment 4 : SELECT * FROM projects WHERE M4
    - Fragment 5 : SELECT * FROM projects WHERE M5
    - Fragment 6 : SELECT * FROM projects WHERE M6

- Allocating the fragments according to branch location
    - Site 1 (Nairobi) => Fragment 1 and Fragment 2
    - Site 1 (Nakuru) => Fragment 3 and Fragment 4
    - Site 1 (Kericho) => Fragment 5 and Fragment 6

- To preserve database integrity on synching foreign keys, we perform derived fragmentation on all tables that are linked to the projects table

    Performing derived horizontal fragmentation on Assignations Table.
    Nairobi Site :
    - Assignations SEMI-JOIN Fragment 1
    - Assignations SEMI-JOIN Fragment 2

    Nakuru Site :
    - Assignations SEMI-JOIN Fragment 3
    - Assignations SEMI-JOIN Fragment 4

    Kericho Site :
    - Assignations SEMI-JOIN Fragment 5
    - Assignations SEMI-JOIN Fragment 6

Reconstruction and Execution of Inclusive Queries

Full reconstruction of the whole schema may happen after a software failure. Partial reconstruction may happen when an inclusive query gets executed. An inclusive query is a query that involves many fragments.
For example ;

- App 1 is an inclusive query because it requests data from all the 6 fragments of the projects table located in all 3 sites.
- App 2 is NOT an inclusive query because it only requests data from one site.

Full Recontruction:

Full Reconstruction is implemented as a function that performs unions on all the fragments distributed in the different sites for each global relation. Below is the pseudocode and a snippet of the actual reconstruction function :

Full_reconstruction :
Create reconstruct schema at site 1
Connect to all sites

- Reconstruct projects table :
    - Reconstructed table = (Union all fragments in Site 1) union (Union all fragments in Site 2) Union (Union all fragments in Site 3)
- Reconstruct job descriptions table
    - Reconstructed job descriptions = (Union all fragments in Site 1) union (Union all fragments in Site 2) Union (Union all fragments in Site 3)
- Reconstruct employees table :
    - Reconstructed employees table = (Union all fragments in Site 1) union (Union all fragments in Site 2) Union (Union all fragments in Site 3)
- Reconstruct assignations table :
    - Reconstructed assignations table = (Union all fragments in Site 1) union (Union all fragments in Site 2) Union (Union all fragments in Site 3)

Store all the reconstructed tables in the global schema

Execution of inclusive queries

App 1 is a query that affects all 6 fragments of the projects Table. Below is its pseodocode

- Connect to the databases on all sites
- Execute the following lines on each site ;
    - Execute app 1 query on the sites project fragments
    - Perform a union on the results and store the results as an object
    - Return the union results to the main site
- Perform a union on all the returned site objects
- Display the union results as a table

Communication between the servers.

Communication between the different servers was achieved by :

1. Ensuring that all servers can ping each other. This was achieved by placing all sites in the same network.

2. Using two python libraries that acted as connectors between the python-client application and the three different database platforms.
    - Psycopg2 python library acted as a connector between python and PostgreSQL database.
    - Mysql-connector-python acted as a connector between python and both MySQL and MariaDB databases.