# CS631 : IMPLEMENTATION TECHNIQUES FOR DBMS

## COURSE PROJECT REPORT

**Topic : Distributed Caching in PostgreSQL**


**Team Members**

**Kumaran (23M0803)**
**Chaitra (23M0831)**
**Harikrishna (22M0759)**

***Saturday, 02 December, 2023***

**Problem Statement :**
Caching of remote data can lead to significant performance improvements. In this project, you will maintain a relation on one instance A of PostgreSQL as a cached of a relation oon instance B. A stores only a subset of data that is accessed locally, and when a query needs data that is not available at A, the data is fetched from B and cached. Your goal is to do caching on a per-key basis, so if data for a key is not available, it is transparently fetched from the backend, with upper layers not aware of it. You could implement this by adding support for local caching of data for a foreign table, or you can implement it on your own without using foreign tables.

**Creating a Remote Server :**

- Modify the PostgreSQL configuration file.

$ sudo vi /etc/postgresql/12/main/postgresql.conf
Uncomment the line *listen_addresses = 'localhost'*

- Modify the pg_hba.conf file.

# IPv4 local connections:
host all all 0.0.0.0/0 md5

- Allow port through the firewall.

$ sudo ufw allow 5432/tcp

**Connecting to Foreign Database :**

We used Foreign Data Wrapper (FDW) of PostgreSQL to connect to a remote database. This wrapper internally uses the *libpq* library to make connections. However, making this connection is much easier through direct commands from psql.

```
create extension postgres_fdw;

CREATE SERVER <foreign server name>
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (host '<ip address>', port '<port no>', dbname '<foreign dbname>');

CREATE USER MAPPING FOR <foreign username>
SERVER <foreign server name>
OPTIONS (user '<username>', password '<password>');

create foreign table <local table>
       (
       // columns of the table
       )
       server <foreign server name> options (table_name '<foreign table name>')
```

Now, the command *select * from <local table>* will directly fetch data from the foreign table.

**Approach :**

- Create a <cached local table> with the exact same schema as <local table>, which is the local copy of the foreign table.

```
CREATE TABLE <cached local table>
(
// columns of the table
)
```

- For every SELECT query on the <local table>, we redirect it to the <cached local table> to check if the primary keys are available. Create a new function char *replaceWord() to replace the table name from <local table> to <cached local table>.

```
char* replaceWord(const char* s, const char* oldW,
                  const char* newW)
{
    char* result;
    int i, cnt = 0;
    int newWlen = strlen(newW);
    int oldWlen = strlen(oldW);

    // Counting the number of times old word
    // occur in the string
    for (i = 0; s[i] != '\0'; i++) {
        if (strstr(&s[i], oldW) == &s[i]) {
            cnt++;

            // Jumping to index after the old word.
            i += oldWlen - 1;
        }
    }

    // Making new string of enough length
    result = (char*)malloc(i + cnt * (newWlen - oldWlen) + 1);

    i = 0;
    while (*s) {
        // compare the substring with the result
        if (strstr(s, oldW) == s) {
            strcpy(&result[i], newW);
            i += newWlen;
            s += oldWlen;
        }
        else
            result[i++] = *s++;
    }

    result[i] = '\0';
    return result;
}
```

- Modify the function exec_simple_query() in the file postgres.c to modify the query.

```
char token[100];
int i, j = 0,k=0;
int str_len = strlen(query_string);
for (i = 0; i < str_len; i++) {
	if (query_string[i] != ' ') {
		token[j++] = query_string[i];
	} else {
		token[j] = '\0';
		if(k==3)
			break;
		j = 0;
		k+=1;
	}
}
if(strcmp(token,"<local table>")==0) {
	char new_query[1200];
	strcpy(backup_query,query_string);
	strcpy(new_query,query_string);
	char* final = replaceWord(new_query,"<local table>", "<cached local table>");
	strcpy(query_string, final);
}
```

- In the exec_simple_query() function, where the PortalRun() function is called, create a new variable ‘localnotfound’ which checks the return value of the PortalRun() function. The PortalRun() function sets the value of ‘localnotfound’ to 1 if the primary key is not found in the <cached local table>

```
int localnotfound=0;
(void) PortalRun(portal, FETCH_ALL, true, true, receiver, receiver, &qc, &localnotfound);
if(localnotfound) {
	char var[] = "INSERT INTO cached_student ";
	strcat(var,backup_query);
	strcat(var,";");
	char* final = replaceWord(backup_query, "localOfRemote", "cached_student");
	strcat(var,final);
	strcat(var,";");
	exec_simple_query(var);
}
```

- Modify the PortalRun() function in the file pgquery.c to include the variable ‘localnotfound’.

```
bool PortalRun(Portal portal, long count, bool isTopLevel, bool run_once, DestReceiver *dest,
DestReceiver *altdest, QueryCompletion *qc, int* localnotfound) {
	// some code start
	if(nprocessed <=0 )
		*localnotfound=1;
	// some code end
}
```

- Modify the function declaration of PortalRun() in pgquery.h file.

```
extern bool PortalRun(Portal portal, long count, bool isTopLevel, bool run_once, DestReceiver
*dest, DestReceiver *altdest, QueryCompletion *qc, int* localnotfound);
```

**Summary of the Approach :**

- Redirect all SELECT queries from the foreign table to the cached local table.
- If the tuples returned from the cached local table are zero, fetch the tuples from foreign table.
- Insert the fetched tuples into the cached local table for future. Display the inserted tuple

**Conclusion and Future Work :**

- We have implemented this approach only for SELECT queries, which can be extended for more complicated queries in the future.
- The connection to the foreign database is refreshed every time the tuples are fetched from the foreign table. This can be improved by not letting the connection refresh when fetched from the foreign table.

**********