Addis Ababa University

Collage of Natural Science

Development of Amharic Locale Integrating System in Open Source Database

Yonathan Misgan Mengie

A Project Submitted to the Department of Computer Science in Partial Fulfillment for the Degree of Master of Science in Computer Science

Addis Ababa, Ethiopia

June, 2020

Addis Ababa University

College of Natural Sciences

Department of Computer Science

Yonathan Misgan Mengie

Advisor: Dr. Solomon Atnafu

This is to certify that the Project prepared by Yonathan Misgan, titled: ***Development of Amharic Locale Integrating System in Open Source Database*** and submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the Examining Committee:

	Name	Signature	Date
Advisor:	Dr. Solomon Atnafu	_____	_____
Examiner:	Dr. Yaregal Assabie	_____	_____
Examiner:	Dr. Solomon Gizaw	_____	_____

Abstract

Nowadays, in many domains, managing data according to local convention is required to be taken into account. Due to this, most of the existing system environments include the models for internationalization to allow developer to easily develop software that support different locales. Even though, software's include models of internationalization the localization work conducted on Amharic locale development has limitations which has a considerable impact on managing Amharic locale data in database as well as in other software system. There are attempts to support Amharic locale in database and other software by developing locale schema in XML however, these works are not including all the required features of Amharic locale. Calendar and collation sequence are examples of Amharic locales which are not managed in the current database software environment. For supporting Amharic locale in a database system we developed an extension to meet needs of Amharic locales. In this report, we present our work that extends the PostgreSQL database system to manage Amharic locale data. We introduce new data types along with a set of functions to support Amharic locale data in PostgreSQL database.

Finally, for integrating the extension into PostgreSQL database we developed source code in C programing language, control and SQL script file which contain real implementation of the locale, information about the extension and map all the new data type, functions and operators to SQL command respectively. Testing of the extended system produced the expected result.

Keywords: database management system, database extension, Amharic locale, locale in PostgreSQL.

Acknowledgement

First and foremost, I would like to express my deepest gratitude to my advisor, Dr. Solomon Atnafu, for his time, helping me in getting this title and supported me throughout my project with his patience and knowledge. He offered invaluable advice, support and guidance. This project would not have been possible without his constructive comments on every aspect of the study.

Also, I would also like to thank Debre Tabor University and AAU for giving this chance. Finally, I would like to thank my family, friends, colleagues and all other peoples who have contributed in one way or another on this work.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Abbreviation and Acronyms

Alt	Alternative
CLDR	Common Locale Data Repository
CMS	Content Management Software
DB	Database
DBMS	Database Management System
DDL	Data Definition Language
DLL	Dynamic Loadable Library
DML	Data Manipulation Language
ETB	Ethiopian Birr
GLIBC	GNU C Library
ISO	International Organization for Standard
L10N	Localization
NLS	National Language Support
OSS	Open Source Software
PGXS	Extension Building Infrastructure
PGXN	Postgres Extension Network
POESSA	Private Organizations' Employees Social Security Agency
SO	Shared Object
SQL	Structured Query Language
UC	Use Case
VAT	Value Added Tax

Chapter One – Introduction

Technology lets us communicate across international boundaries. That's one of the best things about it, both from the developer's perspective and the user's. For software developers, globalization means a potentially unlimited market for any new program or application. But it's not enough for a website or program to support multiple languages. It has to make sense for users in every country that it reaches. Otherwise, it's not truly global it is just multilingual. Due to this, software internationalization and localization are important in distributing and deploying software to different regions of the world.

Internationalization is the process of designing a software so that it can be adapted to various languages and cultural conventions without engineering changes [1, 2]. There are different aspects to be consider during internationalization. The first is designing and developing the software in such a way so as to remove barriers to localization. This includes enabling the use of Unicode or ensuring the proper handling of legacy character encodings where appropriate, taking care of the concatenation of strings and avoiding dependence of user interface string values within a code. The second one is providing support for features that may not be used until localization occurs. For example, adding markup to support bidirectional text. The third aspect is enabling software to support local and other culturally related preferences. Typically, this involves incorporating predefined localization data and features derived from existing libraries or user preferences. For examples date and time formats, local calendars, number formats, numeral systems, sorting, handling of personal names and forms of address. The fourth aspect is separating localizable elements from source code or content so that localized alternatives can be loaded or selected based on the user's international preferences as needed. Once everything is done to simplify the process of localization, the process of internationalization may be considered complete [3].

Software localization is the process of adapting internationalized software to culture and language of the end user's locale. such as: writing directions, date and time formats, currency formatting conventions, address formats, input methods, decimal separators, etc. Localization considers culturally sensitive aspects and technical details that go beyond translation. Because in translation once you translate the user interfacing parts of a software program into the target

language and begin to use that software a user will soon notice some usability difficulties. Localization is no longer just an option for software which want to go international, it is now becoming a must. By localizing our software, we can build a bridge to our target users. Some of the importance of localizing a software for international markets are reach new markets fast, customer satisfaction, reduce costs, ensuring better understanding of product functionalities etc. [4, 5].

Locale is a set of parameters used to identify the user's language, country and other preferences. It is the combination of a language and a geographical region with all the cultural implications involved [2, 5, 6].

The Ethiopic (Ge'ez) script is used as the writing system of Semitic language spoken in Ethiopia and it includes more than 400 characters. Although the language (Ge'ez) only to be used in vernacular speech (it now serves a liturgical function only), the script is still widely used for writing the Ethiopian and Eritrean Semitic languages such as Tigré, Amharic and Tigrinya [7, 8, 9].

Similar to the script Ethiopia has its own ancient calendar. The Ethiopian calendar has difference with the Gregorian calendar in terms of day, month and year. The Ethiopic calendar has 13 months, 12 of which are with 30 days each and an additional month at the end of the year with 5 or 6 days depending whether the year is a leap year or not [10]. As a result, a huge amount of data in Ethiopic script use Ethiopian calendar in which it needs to be stored in databases according to Amharic locale system.

1.1 Motivation

Due to the continuous rise of local data in many domains, the question how to manage these data efficiently based on the local requirement of the user has become of crucial importance. However, both internationally and nationally, most of the data management software are developed to support primarily English language. Apart from government efforts, individual developers also need to be motivated to take part in localized content management development process. Furthermore, in countries like Germany, Japan and China, there are also local language support environments along with the English for both proprietary and open-source environments [11, 12]. However, no such environment exits in the Ethiopian context

to support Amharic locales (calendar, collation order, currency, etc…) in the database system, particularly in open-source environments.

Today, open source software are becoming more popular than commercial software all over the world because of its flexibility in supporting localization, as their products are in use increasingly everywhere. Since open source systems are free and open for every user, it attracts more users who have the same interest, to become co-developer for extending, improving and modifying the initial systems [13]. There is a strong need among local organizations to build a software system that support Ethiopic scripts without modifying the executable files. It is therefore necessary to enhance existing open-source data management systems so that they become compliant to local requirements. Such initiatives should also consider the system enhancement and can be easily integrated with the existing systems and facilitate new and better application development.

Many languages in Ethiopia are using Ethiopic script, Ethiopian calendar, currency, and other conventions of Ethiopian culture. This shows software systems and database systems need to be able to understand and present the correct information to the user without the need for conversion. Similarly, the collation order everyone uses according to the Ethiopian alphabet order. Most data are collected, represented and used according to Ethiopian calendar only a few organizations use the Gregorian calendar, which differ substantially from the local practice. All of these are important aspects to be considered to develop DBMS that can handle multiple inputs of Amharic locales and that can automatically check those inputs to ensure that the right format is used.

The need for locale support nowadays is therefore immense from every aspect, as it avoids confusion and brings benefits to industries, customers, and government, simply to everyone. Even more importantly, storing information in own native language and based on cultural practice is a need in many respects. Research has shown that the lack of availability of information in locale convention is the main reason for the slow growth in Information Communication Technology (ICT) sector by most of the developing countries. Developing the necessary technique to use locale and making available in association to open source software that can freely be used by any organization is critically important.

1.2 Statement of Problem

Usually, programs are culturally neutral it consists of interaction between many components which are arranged in specific patterns and perform tasks. Using a common language is quite handy for communication between developers, maintainers and users from all countries. However, as soon as we add linguistic and cultural specificity to these programs most people are more comfortable with their own native language, and would prefer to use their locale for day to day's work, as far as possible. Most of the existing system environments include the models for internationalization to allow developer to easily develop software that support different locales [2]. Even though, software's include models of internationalization the localization work conducted on Amharic locale development has limitations which has a considerable impact on managing Amharic locale data in database as well as in other software system. Calendar and collation sequence are examples of Amharic locales which are not managed in the current software environment.

A calendar is a human abstraction of the physical timeline [11]. A calendar can measure time using any well-defined time unit. A typical calendar property is the language in which time values are expressed. For example, in the Ethiopian calendar Amharic and Geez are used to express periods of time, but in the Gregorian calendar English is used to express periods of time in the United States. The usage of a calendar depends on the cultural, legal, and even business orientation of the user. Most of the existing operating systems use Gregorian calendar system. However, most of the data in Ethiopia are available with Ethiopian calendar system and users in Ethiopia are not comfortable with the Gregorian calendar since they use Ethiopian calendar in their day-to-day activities. This usually create inconvenience when the user wants to have a reference to Ethiopic date as they had Gregorian calendar at their operating system. An example query to demonstrate this is given below.

- *Q1: Select current_date*;

Q1 returns '2020-5-20' where the current year in Ethiopian calendar is '2012',

Also the existing database management systems is not capable of supporting time values Ethiopian calendar.

- *Q2: insert into testable values ('5/13/2012');*

- *Q3: insert into testable values ('ሰኔ 21 2012');*
- *Q4: insert into testable values ('ሰኔ ፭ ፳፻፲፪');*

Q2 returns an error message because the GC have only ‘12’ month per a year.

Q3 and *Q4* returns an error message because the GC use English to express periods of time values.

Therefore, the user need to convert the Ethiopian calendar to Gregorian calendar using different calendar conversion software in order to manage Ethiopian calendar in a database.

Sorting and searching are the main important operations in database system as well as in other computer systems for efficient management of data. However, texts written using Ethiopic script are sorted in a database is based on compatible equivalence. Compatibility equivalence is type of equivalence between characters or sequences of characters represent essentially the same text but might have different code point representation. For instance, if we have a database table *Student* with column f*name* values recorded as ሐመልማል and ለገሰ and querying like:

- *Q8: Select name from Student order by fname asc;*

Q8 returns ‘ሐመለማል, ለገሰ’. Which is incorrect collation order according to Amharic language that uses Ethiopic alphabetic order.

To the knowledge of us localization is potentially performed multiple times, for different locales uses the infrastructure or flexibility provided by internationalization as an integral part of ongoing development. This work is intended to address the problem of integrating Amharic locale in open source database environment.

1.3 Objective

General objective

The general objective of this work is to develop Amharic locale and integrate in open source database.

Specific objectives

- Explore related works in locale integration in database environment.
- Identify different techniques and methods that have been used in the area of database localization.
- Identify the DBMS component that are linked to locale integration.
- Identify the requirement of Amharic locale integration in database systems.
- Design an appropriate Amharic locale integrating system.
- Design a prototype using a selected open source database.
- Test the prototype.

1.4 Methods

In order to successfully accomplish the objectives, the following methodologies will be used.

- **Literature Review**

Literature review will be conducted to acquire enough understanding of the various techniques of localization. Specifically, literatures in the area of DBMS component localization (concepts and approaches) will reviewed. Reviewing these literatures and related works help to gain the required knowledge on the area and assists in identifying the right methods and tools for implementing the different components of the system.

- **Data collection**

Data on the practice of using language components related to locale will be collected and analyzed. This will be done particularly considering Amharic language assuming that all the other local languages do have the same locale.

- **Implementation**

During implementation, the convention of Amharic locale will be code into schema, which includes implementing the attributes and methods of each object and integrating all the objects such that they function as a single package.

- **Testing and Evaluation**

The developed system will be tested and evaluated for effectiveness.

1.5 Scope and Limitations

This project work will be conducted to explore the advantage of developing and using Amharic locale integrating system for database. Even though the work will solve problems of data management with Amharic locales, it does not take into consideration to localize the DBMS user interface (menus, buttons, captions, labels and dialog boxes etc...), SQL query and logo.

1.6 Application of Results

The result of this work will be important for database developers and users of database in local language. The possible importance of the proposed work will enable to:

- easily manage data formatted to suit Amharic locales system (such as collating sequence, calendar and date/time etc…);
- reduce costs for developers since they can focus only on handling Amharic locale system instead of dealing with conversions after developing the whole DBMS in English locale;
- local language development in the digital world and to better understanding of product functionalities of DBMS; and
- use as an input for other works.

1.7 Organization of the Document

The rest of this report is organized as follows. The second chapter presents the literature review conducted to share opportunities, standards, best practices, and experiences on localization and software extension. The third chapter describes the related work done before both on Amharic and other locale on database and other software environment. The fourth chapter describes the system analysis that includes functional, nonfunctional, and system models. The fifth chapter presents integration of Amharic locale in database that includes system architecture, the qualities that need the attention of the developer and hardware/software mapping of the system. The sixth chapter presents implementation of the project. Finally, conclusion and future direction is given in chapter seven.

Chapter Two – Literature Review

This chapter reviews the literature on available open-source software, database management system, software extension, localization and some of Amharic locales are presented. The review helps to understand the problem of in the use of national locale and to explore the domain for possible appropriate solution.

2.1 Open Source software

In recent years, Open Source Software has become a major interest both for software industry and for economic theory [14]. Open source software is free to run the program, for any purpose. They are free to inspect the actual source code of the program to determine how it works. They are free to modify and adapt the software to specific needs; to redistribute copies to whomever; to release code improvements to the public, and to benefit the whole community [13, 15].

Motivations and some practical advantages of using Open Source Software are:

- The availability of the source code and the right to modify is very important. It enables unlimited tuning and improvement of a software product. It also makes possible to port the code to new hardware, to adapt it changing conditions, and to reach a detailed understanding how the system works. The right to modify and make improvements to the code permits all the advantages due to the modifiability of the software to be shared by large communities [13].
- The right to use the OSS in any way. A large number of users, which help in turn to build up a market for support and customization of the software, which can only attract more and more developers to work in the project. As more developers look and work on the common source code and more rapid innovation [16].
- There is no one with the power to restrict in a unilateral way how the OSS is used, even in a retroactive way. For instance, when a proprietary software vendor decides not to upgrade some software product for some old platform. In this case, users can only stick to the old version of the software, or switch to another product. If OSS is

used, users can also fund some development for the desired platform, or look for other vendors to provide the upgrades (of the very same product) [17, 18].

- Local users do not need to learn English for using the open source software. Localization of applications can be prioritized according to the national needs [19].

Open source Database Management System

A Database Management System is the software that enables users to define, create, maintain and control access to the database. We can also define as a software that interacts with the user's application program and the database. Typically, a DBMS provides the following facilities: It allows users to define the database, through a data definition Language (DDL). The DDL allows users to specify the data types and constraints on the data to be stored in the database. It allows user to insert, update, delete and retrieve data from a database, usually through a data manipulation language (DML). Having a central repository for all data and descriptions allow the DML to provide a general enquiry facility to this data by using a query language. The most common query language is the structured query language. DBMS provides a controlled access to the database [20]

For example, the following functions provided by the DBMS:

- A security system, which prevents unauthorized users accessing the database and prevent unauthorized transactions.
- An integrity system, which maintains the consistency and validity of stored data.
- A concurrency control system, which allows multiple users to access the database simultaneously.
- A recovery control system, which provides a method for backing up and restoring the data in a database, in the event of some type of system failure.
- A user accessible catalog, which contains descriptions of the data in the database etc.

Open-source database is database software for which the source code is available for anyone to inspect, modify and enhance. The software usually includes a license for programmers to change the software in any way they choose. They can fix bugs, improve functions, or adapt

the software to suit their own needs [21]. According to reports of DB-Engines [22], an online initiative to collect and present information on database management systems, popularity of open source databases are growing faster than commercial databases.

Some Popular Open Source DBMS

Some Examples of the most popular free or open-source database management systems are: MySQL and PostgreSQL.

MySQL: MySQL is the world's most popular open source database software. The industry leaders like Yahoo, Google, Nokia, YouTube and Wikipedia are using it. MySQL enterprise edition includes a comprehensive set of advanced features, management tools and technical support to achieve high levels of MySQL scalability, security, reliability and uptime. MySQL is primarily an RDBMS and ships with no GUI tools to administer MySQL databases or manage data contained in a database. For those who want to use GUI, MySQL Workbench is available as a free integrated environment, which enables users to graphically administer MySQL databases and visually design database structures. MySQL Workbench is available in two editions: the regular free and Open Source Community Edition, which may be downloaded from the MySQL website, and the proprietary Standard Edition, which extends and improves the feature set of the Community edition.

PostgreSQL: The object-relational database management system now known as PostgreSQL is derived from the POSTGRES package written at the University of California at Berkeley. It is developed by PostgreSQL Global Development Group. PostgreSQL is available for all platforms Mac, Windows, Solaris and Linux under MIT license. PostgreSQL supports all the properties of major databases. It has more than 15 years of active development and a proven architecture that has earned a strong reputation for reliability, data integrity, and correctness. Compared with the other open source database system, the extensible functionalities and capabilities of PostgreSQL attracted great interests from database administrators and software developers.

2.2 Software Localization

The aim of localization is to give a product look and feel having been created specifically for a target market, no matter their language, culture, or location [23]. This process involves the

translation of all native language strings into the targeted languages as well as customizes the graphical user interface (GUI) in order to make it relevant to the targeted market. In addition, the process of localization include [2, 12, 23]:

- Modifying content to suit the tastes and consumption habits of other markets
- Adapting the design and layout to properly display translated text
- Adapting to the local requirements (such as currencies, date and units of measure)
- Using proper local formats for dates, addresses, and phone numbers
- Addressing local regulations and legal requirements

Therefore, Software localization is not limited to "translating applications menus, dialog boxes, alert boxes and content areas into a language or regional dialect" as defined in the human computer interface guidelines [12].

Locales: a set of parameters used to identify the user's language, country and other preferences. It is the combination of a language and a geographical region with all the cultural implications involved [24]. To support users in different cultures, programs must not only use translated text. However, also be adapted to local conventions. These conventions differ by language or region and include the formatting of numbers, dates, times, and currency values, as well as support for differences in measurement units, text sorting order, and other data formats and services [2, 5, 12]. A locale is divided into categories, which define different parts of the whole locale. The categories of locale defined in <*locale.h*> are as follows:

Table 2.1: *List of locale categories*

Locale category	**Locales**
LC_ALL	Affects all the categories at once
LC_COLLATE	String sort order
LC_CTYPE	Affect character classification
LC_MONETARY	This category applies to formatting monetary values
LC_NUMERIC	Contains nonmonetary numeric formatting items
LC_TIME	Applies to formatting locale-specific date and time, pointers to month and weekday names, and a.m. and p.m. designation;
LC_MESSAGES	Applies to the Language of messages

Database System Localization

A database localization is very common task for software development process. Many applications use databases to store language and culture specific data. This approach is extremely useful when developing web and enterprise applications [25, 26].

Localization of Content Management

A content management system is a computer application that supports the creation and mange of digital content. It typically supports multiple users in a collaborative environment. Most CMSs are web solutions designed to help develop and maintain shared information, usually for websites but also for document and content control. Most current CMSs work with content in XML format, separating content from format (design elements), allowing numerous output combinations with less work. CMS solutions are also intended to empower multiple users to author, edit and share information in multiple formats [27].

2.3 Amharic Locales

Calendar: A calendar is a system of organizing days for social, religious, commercial or administrative purposes. It can measure time using well defined time unit. This is done by giving names to periods of time, typically days, weeks, months and years. A date is the designation of a single, specific day within such a system [10, 11].

The Gregorian calendar is the most widely used calendar in the world today specially for database and other computer system. It is the calendar used in the international standard for representation of dates and times: ISO 8601:2004. It is a solar calendar based on a 365 days' common year divided into 12 months of irregular lengths 11 of the months have either 30 or 31 days, while the second month, February, has only 28 days during the common year. However, nearly every four years is a leap year, when one extra or intercalary day is added on February. Making the leap year in the Gregorian calendar 366 days long. The days of the year in the calendar are divided into 7 days' weeks. The international standard is to start the week on Monday. However, several countries, including the Ethiopia, US and Canada, count Sunday as the first day of the week [10, 11].

Ethiopia has its own ancient calendar which is 7 years behind the Gregorian calendar, from September 11/12 to December and 8 years behind for the rest. The Ethiopian calendar has

difference with the Gregorian calendar in terms of day, month and year. In the Ethiopian calendar the new year starts on Meskerem(መስከረም) 1 which falls on 11th or 12th of September in the Gregorian calendar depending on the Ethiopian leap year. The Ethiopian calendar is based on the Coptic calendar, although it differs with regard to names of months and days which is language specific. Like the Coptic calendar, the Ethiopic calendar has 12 months of 30 days plus 5 or 6 additional days depending on the leap year, which comprise a thirteenth month [10, 11]. Maybe the clearest example of the different ways culture views around the world is demonstrated by our perceptions of the non-tangible entity of time.

Date and Time: time is important aspects of all real-world phenomena. Events occur at specific points in time; object and relationships among object exist over time. The ability to support this dimension of real world in a database is essential in many computer applications. Such as banking, inventory control, medical records and geographical information system.

Ethiopia shares the 24-hour day convention with the rest of the world but differs on when the day begins. The two-part day division around mid-day (AM for anti-meridian and PM for post-meridian) is also a foreign notion taken for universal in localization systems. Where the "<am>" and "<pm>" day divisions of Amharic translation is an approximation that is no more than serviceable [12]:

<am>ጠዋት</am> and <pm>ከሰዓት</pm>

While these translations could be understood under the context of the foreign conventions that they map, they are not ideal for Ethiopia. Naturally, Ethiopia will want to apply its own conventions that are already millennium old. ጠዋት, ረፋድ, እኩለ ቀን, እኩለ ሌሊት some of the examples of day division in Ethiopia.

Ethiopia does not have a well-established preference for digital time formats in database system and other computer systems, but we want to establish computerized systems into a society. An example digital time format under United States English conventions appears as:

Mon 27 Feb 2018 12:00:00 PM EAT

The equivalent date and time under the Ethiopian Amharic convention as available on Linux systems today appears as:

ማክሰ ፌብሩ 26 ቀን 12：00：00 ከሰዓት EAT 2018 ዓ/ም

This represents a loose mapping of some Amharic conventions onto an external reckoning of time. This is only translation and not localization in its truest sense. A hypothetical Ethiopic date and time presentation might look as:

ማክሰኞ፣ የካቲት 19 ቀን 6：00：00 እኩለ ቀን 2010 ዓ/ም or

ማክሰኞ፣ የካቲት 19 ቀን 6：00：00 እኩለ ቀን፳፻፲ ዓ/ም

Numbers: Arabic numbers are used for computational purpose while Ge'ez numbers are needed for Non-computational purpose of data manipulation like calendar. Ethiopic digit glyphs are derived from the Greek alphabet, possibly borrowed from Coptic letter forms. The Ethiopic number system represents 1-9 by symbols but does not have zero. However, there are independent symbols for 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 like Greek and Coptic. After 100 the Copts and Greeks define alphabetic letter to represent multiple of hundred up to 900 but in Ethiopic numerals there is no new numeral glyph until 10000 rather chose to prefix ones and tens to 100. In modern use, Arabic numerals are often used in Ethiopia [7].

Currency: is a system of money (monetary units) in common use, especially in a nation. Ethiopian birr (ብር), US dollar, British pound and European euro are examples of currency. These various currencies are recognized stores of value and traded between nations in foreign exchange markets, which determine the relative values of the different currencies [28]. Ethiopian currency is defined by government and has limited boundaries of acceptance. ብር is characters used to denote that a number is a monetary value in Ethiopia.

2.4 Software Extension

OSS are developed and produced to adapt new features by separating the software into modules (procedures, objects, components, etc.). However, all these decomposition approaches are mostly applied at the source code level. When it comes to compilation, the different modules of an application are often still linked to a single executable. When changing a module or new functionality is required, the change is in source code but the whole application has to be re-compiled or at least re-linked. This fact makes it hard to enhance software systems especially with new modules that are developed by third parties. As a result,

extensions are able to overcome this problem. By developing extension without modifying the source code, we can enhance the software.

In computing, extension is a software component that adds a specific feature to an already working standalone computer program. When a program supports extension, it enables customization. An extension itself is not independently functional it needs other software with which to work. Extensions are simply files that work in combination with software to achieve a certain task that neither the extension nor the software can provide individually. It has some more instructions that the base program does not have. Either the base program or the extension can tell those instructions to run. The base program can incorporate the extension code into itself through dynamic loading. That is, the user can specify an object code file (e.g., in form of shared library or DLL) that implements a new functionality and will load it as required at run time [29].

Applications support extensions for many reasons. Some of the main reasons include to:

- enable third-party developers to create abilities which extend an application,
- support easily adding new features,
- reduce the size of an application,
- integrate a software module into an existing software system without changing existing source code.

In addition to that, it should be also pointed out that localize the database management system to support the target locale is not flexible and once a database is created we cannot change or add other locales for that database anymore [30]. But, with extension we can mix rules from one locale with other locales even in the same tables for different columns.

Extending PostgreSQL

PostgreSQL provides a pluggable architecture that allows us to extend the functionality of DBMS without rewriting its engine using extensions. An extension is loaded into a running PostgreSQL server process as needed. If we don't need use the extension, it will not be loaded. Once we load it into the database, this extension can function as built-in features. The extension is created in the form of a dynamically loadable object module. This extensions module consists of a dynamically loadable library (shared object), configuration (control) file

and SQL files(script). Packaging all these files as an extension gives the benefit that PostgreSQL will recognize them as a single unit and it is helpful to simplify database management. There are two phases to add an extension module to the PostgreSQL server. First, we create the extension in C language and then compiling it into a dynamic object module (.dll or .so). Next, load the extension to PostgreSQL server by *CREATE EXTENSION* command [30, 31].

There are four basic file types that are required for create an extension for PostgreSQL:

- C Code
- Control File
- SQL File
- Makefile

C Code

C code is real implementation of extension. In this C code we have C implementations of functions, types and operators [30, 31].

Control file

It specifies properties/metadata about the extension, which tells the basics about extension to PostgreSQL, and must be placed in the installation's SHAREDIR/extension directory. The file format must be *extension_name.control*, Parameters inside control files must follow the same convention as postgresql.conf file (i.e. *parameter_name = parameter_value*).A control file contain information about [30, 31]:

- The directory containing the extension's SQL script file.
- The default version of the extension.
- A comment (any string) about the extension.
- The character set encoding used by the script file.
- A list of names of extensions that this extension depends on, and other properties of the extension

The script file

SQL script is mapping file, which we used to map all the PostgreSQL function with the corresponding C function. It contains any SQL commands about functions, types and operators we want to add as an extension on PostgreSQL. It also includes the required DDL and DML operations of extension. This file should be of format extension—version.sql [30].

Makefile

Postgres provides an infrastructure called PGXS for building the extensions against installed Postgres server. To use the PGXS infrastructure for our extension, we must write a *makefile*. Makefile provides a way to compile our C code because PostgreSQL will not compile a C code automatically. In the makefile, we need to set some variables and include the global PGXS makefile [30].

Table 2.2: *Some of the PostgreSQL extension Makefile variables*

Name	**Description**
MODULES	List of shared-library objects to be built from the C code.
EXTENSION	Extension name
PG_CONFIG	Path to *pg_config* program for the PostgreSQL installation to build against
MODULEDIR	Subdirectory of *prefix/share* into which DATA and DOCS files should be installed (default is extension)
DATA	random files to install into prefix/share/$MODULEDIR

Chapter Three – Related Work

In this Section, related works conducted for both Amharic and non-Amharic locales on database and other software are discussed. First, the reviews on the existing works on the area of database localization will be presented. Then, the Amharic localization for other software are presented.

3.1 Database Localization

Different works have been done for database system localization in different cultural settings. Each of these works has their own strong and weak parts. In this section, some of those works that have been done so far for database system localizations are reviewed.

Oracle Database Localization

According to Bhatiya [32] oracle support, globalization as National Language Support (NLS) features to choose a national language and that enables you to manage data in users' native languages and locale preferences. It ensures that database utilities, error messages, sort order, and date, time, monetary, numeric, and calendar conventions automatically adapt to any native language and locale convention. These features are implemented with the NLS runtime library. NLS runtime library provides a broad set of language-independent functions that perform proper data management and language-convention manipulations. Activities of these functions for a specific locale preference is governed by a set of locale-specific data that is identified and loaded at runtime. The locale-specific data is structured as independent sets of data for each locale that oracle database supports. This design has its own advantages:

1. To manage memory consumption by choosing the set of locales that you need.
2. To add and customize specific locale without affecting other locales.

Oracle locale builder offers an easy way to customize locale data in oracle database. It provides a graphical user interface through which one can easily view, modify, and define locale-specific data. It extracts data from the text and binary definition files and presents them in a readable format so that you manage the data without worrying about the formats used in these files. The oracle locale builder manages four types of locale definitions in oracle database: language, territory, character set, and linguistic sort. It also supports user-defined characters and customized linguistic rules [32].

The other way to customize set of locale preferences in oracle database is using hardcoded external files (locale data) that are automatically loaded when the locale functions are executed. Locale data is first defined in a text file. The text file must be converted into binary format. You can use the NLS runtime library to convert the text definition file into binary format. After the binary files have been generated, they are automatically loaded during locale function execution [32]. Some of the locales supported by oracle database are English, Dutch, French, Arabic. Amharic and German etc…The main drawback of oracle database globalization is the commercial feature of the database. The vendor needs license cost and additional charge that is significantly recurring fees for supporting additional feature and updates. Since, the source code is guarded secret no one can be solving locale specific problems by their own.

PostgreSQL Database Localization

PostgreSQL Global Development Group [30], presented different locale systems supported by PostgreSQL systems for managing local data efficiently. PostgreSQL manages different cultural and linguistic convention (locale data) by using locale features of the host operating system. Locale support is automatically set when a database cluster is created using initdb. Initdb will initialize the database cluster with the locale features of the operating system (execution environment) by default to provide locale-specific number formatting, translated messages, calendar and other aspects. If you want to use a different locale you can instruct initdb exactly which locale to use by specifying the --locale option. Some of the locales that are supported by PostgreSQL are English (US), German, and Sweden, French Canadian, etc. Initdb creates a new PostgreSQL database cluster. A database cluster is a collection of databases that are managed by a single server instance [30].

For example, for UNIX systems sets the locale to Swedish (sv) as spoken in Sweden (SE):

initdb --locale=sv_SE.

However, when we are using the OS locale feature we can use different settings for different databases. However, once a database is created we cannot change them for that database anymore. They affect the convention of locale preferences.

Even though most of the customization of different locales data in a database system and other software are often similar but the cultural conventions and linguistic factors (locale development) are different. The PostgreSQL development group is not conducted any work that address the issues of Amharic locale. For example, PostgreSQL supports the full set of SQL date and time types but Dates are counted according to the Gregorian calendar because the internal representation of PostgreSQL is as in ISO 8601[1]. Therefore, there is a need to develop locale-integrating system for DBMS that support Amharic locales, which attracts users based on common language or shared racial, gender, ethnic, or nationality-based identities for incorporating the cultural attributes.

MySQL Database Localization

According to Axmark and Widenius [33], different locales need to be managed differently. For this reason, MySQL included model of internationalization and localization for adapting different cultural and linguistic conventions for efficient data management. MySQL support different character set for SQL statement, languages for error messages, locale time at different levels. E.g. at the server, database, table, and column level. For adding, a new locales set to MySQL the procedure depends on whether the locale is simple or complex:

1. If the character set does not need special string collating routines for sorting and does not need multibyte character support, it is simple.
2. If the character set needs either of those features, it is complex.

To customize the locale data in MySQL database [33]:

- To add locale on MySQL, choose the component that you want to add.
- Add a locale element to the MYSET file by using the existing contents in the file as a guide to adding new contents. A simple character set requires only a configuration file, whereas a complex character set requires C source file that defines locale specific function. Example collation functions, character set functions, or both.
- Create a C source file that describes the character set properties and defines the support procedures necessary to properly perform operations on the character set.

[1] *https://www.iso.org/iso-8601-date-and-time-format.html*

- Modify the configuration information. Use the existing configuration information as a guide to adding information for MYSYS.
- Reconfigure, recompile.

Even if it includes model of internationalization Amharic locale is not implemented which is different from other locales in many aspects like calendar and collation sequence.

3.2 Localization of Content Management Software

Ethiopian Localization Project

According to clear IT solution [34], the Ethiopian localization project have modules that extend the functionality of base model to support Amharic locale for VAT[2] customizations. These include names of regional states in Ethiopia, company name, partner name, vat field and vat number with Ethiopic script. The localization work conducted under this project is no more than translation of western conventions into a local language and does not take into consideration the different locales of Amharic like calendar, date/ time format, sorting/ collation order, regular expression etc. However, localization involves much more than the simple translation of text. It has to reflect specific language and cultural preferences in the content, images, overall design and requirements of the software and cover the development of locale model to support Amharic convention.

Localization of Open Source Web Content Management System

According to Rufael Tadesse and Fekade Getahun [27], as part of the localization, a language pack for the CMS is developed to convert the front-end and back-end interface of web development tool Joomla. The language pack is designed in a way to be imported into Joomla and users can easily change the language. In addition, virtual keyboard module is designed for Joomla to enable users to encode Amharic text. The translation component (translation memory) enables users to create Joomla language pack for Amharic, Oromiffa and Tigrigna. Even though the benefits and advantages of translation memory systems are obvious but localization is not about language it involves more than just translation. It includes translation and addresses other cultural aspects. However, this work does not take into consideration

[2] *VAT stands for Value Added Tax*

system level Amharic locales like calendar, date/ time format, sorting/ collation order, regular expression etc.

Website Localization Techniques

The paper [25], presented the analysis and the technique of website localization from the source language (English) to the target language (Arabic). The key points the author considers to localize a website to the target language (Arabic) are resource file and the database table. The author localized and put a separate resource file for the target language. Since the software localization process must include database design adjustments in order to enable the target language supported by the localized software the author uses a commonly used method for developing a multilingual database is to create two tables:

1. The first table contains only the non-language specific fields, such as Primary Key and Boolean values.
2. The second table contains one row per language, which contains the equivalent translation, the language ISO Code and the Foreign Key to reference the primary key from the first table.

This work is conducted at application level and preparing source files for translation in translation memory tools and converting files back adds additional, often time-consuming to the translation process.

Generally, most of these works designed to store data in a database so that the basic functions are provided by the underlying database management system (DBMS). However, most DBMSs use English-like semantics as the query language for data manipulation. Although they can be localized to users' preferred languages, text processing under database systems are still far short some locale developments, such as Amharic locale. Usually, sorting and indexing in a database system use the internal code sequence defined in the given code set. For alphabet-based languages, such as English sorting and indexing is quite straight forward because the internal code sequence naturally reflects the alphabetic order. But, sorting and indexing Ethiopic characters, using Amharic timestamps during processing based on internal code sequence does not work correctly. Therefore, browsing data indexed based on internal code sequence cannot respond to some of the relevant data.

Differences among natural languages have caused data management problem in many computer environments, including databases. Most of these problems are due to the fact that Amharic locale is not integrated in the design of the hardware and system programs. This indicates that there is a need to adapted the existing DBMS to handle data written in Amharic script and to provide a software that can work under Amharic cultural conventions.

3.3 Existing Amharic Locale Implementations

There is a number of locale solutions, external to DBMSs, done by different software developers. Table 3.1 summarizes the most common ones.

Table 3.1: *Existing Amharic Locale Implementations*

Product	**Type**	**Developer**	**Implementation details**	**DBMS Support**
Ethiopian calendar	Application	Many individuals	These are set of calendar applications used to track date, and holidays in Ethiopian calendar. They can also be used to convert dates from and to Gregorian calendar.	No
CLDR	Resource file	Unicode Consortium	A project of the Unicode Consortium to provide locale data in the XML format for use in computer applications. CLDR contains locale-specific information that an operating system can typically provide to applications. CLDR is written in LDML (Locale Data Markup Language).	Yes
Glibc locale data	Resource file	Yeha project (by Daniel Yacob)	The file is developed as part of the GNU C Library and contains locale data. The GNU C Library project provides the core libraries for the GNU system and GNU/Linux systems, as well as many other systems that use Linux as the kernel.	Yes (but only in Linux OS)

The list of available solutions is in fact massive, yet only two of them work with database management system. However, both of the locales are developed as resource file (XML file) for adapting software to the conventions of Amharic language for such common software tasks. Among the types of data, they include translations for:

- language names,
- territory and country names,
- currency unit, including singular/plural modifications,
- weekday, month, era, period of day, in full and abbreviated forms,

- time zones. example cities,
- calendar fields, and
- patterns for formatting/parsing dates or times of the day.

Yet, due to a limitation on Amharic locale development there are some drawback to work according to Amharic locale convention.

Generally, the more likely ways to support locales in the above related works is localization. However, the date/time localization code in different software (for example in PostgreSQL, Ubuntu) use Gregorian calendar logic, even though they can use local language names for them. That means, they allow us to display the French, Germany, Amharic and other names for the Gregorian months, but not a completely different calendar system.

To solve this, we propose that the solution is an implementation using the extension mechanism. Because, the end goal of the work wouldn't need to be part of DBMS itself, but just an extension that anyone can download and install to use Amharic locales conveniently.

Chapter Four – System Analysis

In the previous chapter several works regarding localization of different locales and software extension have been discussed. This provided a base to consider the possible opportunities, limitations, standards, and current status of supporting locale in a database environment. It also indicated the major problems that are holding back development of Amharic locale extension application. Having considered the output of the literature review, requirement elicitation is done to identify the specific requirement for the Ethiopian locale. To identify the requirements document review has been conducted on some database management system tools such as PostgreSQL, MySQL and oracle. The requirement elicitation has shown that most of these database management tools doesn't have a way to support Amharic locale according to Amharic locale convention.

Considering the result of requirement elicitation, the context of the country and expectation of database users, requirements of the proposed system are set.

4.1 Functional Requirements

Functional requirements describe the interactions between the system and its environment independent of its implementation. The environment includes the user and any other external system with which the system interacts. Therefore, the major functional requirements of this project include:

- The extension should allow the user to create a table to store Amharic locale data.
- The extension should allow the user to insert Amharic locale data.
- The extension should allow the user to view the stored data in the database.
- The extension should be able to convert the Ethiopian date/ time to western time format and vice versa.
- The extension should allow the user to modify the recorded data.
- The extension should allow the user to manipulate Amharic locale data
- The extension should allow the user to drop extension.

4.2 Nonfunctional requirements

Nonfunctional requirements specification is done to identify aspects that are visible to the user but not directly related to the functionality of the system. These include constraints on the performance of the system, its documentation, the resources it consumes, and its quality.

Open source: we believe that any software dealing with locale data management can benefit from open source systems. Because, users or system developers may find possible flaws in its implementation. It also enables any interested party to review the code of the extension to modify or add additional functionalities in the implementation and can benefit from code reuse.

Performance: We can develop the extension or plugin in scripting languages, but this may result in fall of performance and complexity of functionality. Most scripting languages are quite a bit slower than optimized C code when executing the same algorithms [30, 31]. If the extension is developed in C, the application will be faster when compared with a plugin developed by most scripting languages.

Availability: The extension should be available any time as long as it is installed.

Error Handling: Since reliability is the major characteristic of high quality system, the extension should be well in every situation. Hence, the extension is expected to handle errors encountered during run time. Errors could arise from users and from the system. Errors that occurred from the wrong doing of users will be handled by appropriate exception handling mechanisms.

Documentation Requirements: The overall development process should be documented including the Requirement Analysis, System Design, and Implementation details.

Platform independence: The extension must be able to run that have a pre-installed PostgreSQL database.

Extensibility: The module will be extensible at any time to enhance and add additional locale features based on the user needs.

Usability: The extension should be easy to use with no additional interface than pre-installed PostgreSQL for Ethiopian locale data management.

4.3 System Models

4.3.1 Use Cases diagram

No system exists in isolation. Every system interacts with human or automated actors that use the system for some purpose, and those actors expect the system to behave in predictable ways. A use case specifies the behavior of a system or a part of a system from an external point of view. A use case is a description of a set of sequences of actions that a system performs to yield an observable result of value to an actor. An actor represents an external entity that interacts with the system. An actor can be a human, a hardware device or even another system that plays with a system. We apply use cases to capture the intended behavior of the system we are developing, without having to specify how that behavior is implemented. The following actor and use cases are identified for the proposed system.

Actor

1. Database user

Use cases

1. Load Extension
2. Create locale table
3. Insert locale data
4. Update locale record
5. View locale data
6. Drop extension

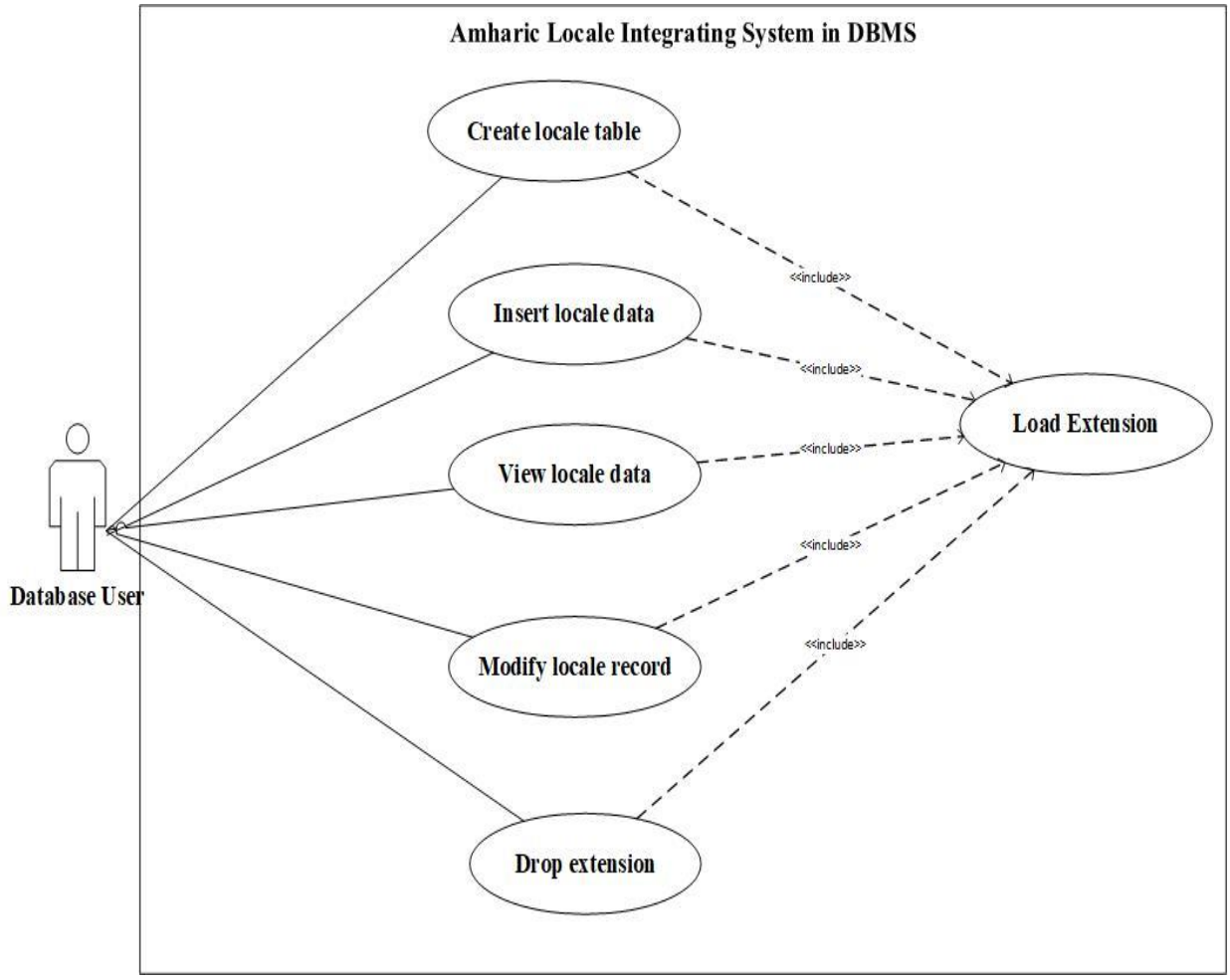

Figure 4.1: *Use case diagram of Amharic locale extension module*

Actor Description

Name: Database user

Description: a database user is a person who is performing different activities of the database.

Use Case Description

The flows of actions to be followed to accomplish the use cases illustrated in Figure 4.1 are described in detail in this section.

Table 4.1: *Use case description for load extension*

Use Case ID	UC-01
Use Case Name	Load extension
Actor	Database user
Purpose	Allow the database user to load Amharic locale extension to PostgreSQL.
Pre-Condition	The actor must add extension.
Flow of Events	1. The actor writes CREATE EXTENSION command. 2. The actor executes the SQL script [Alt 2]. 3. Load the extension objects into the database.
Post-Condition	Conformation " extension loaded"
Alternative Flow	[Alt 2]: If the extension name is not exist. Alt 2.1: The system returns error message. Alt 2.2: The actor writes the correct extension name.

Table 4.2: *Use case description for create table*

Use Case ID	UC-02
Use Case Name	Create table
Actor	Database user
Purpose	Allow the database user to create a database table to store Amharic locale data.
Pre-Condition	The actor must load the extension.
Flow of Events	1. The actor writes a create table query with table column type Ethiopian locale data types. 2. Execute the written query [Alt 2].
Post-Condition	Conformation " table created"
Alternative Flow	[Alt 2]: If the data type is invalid or there is any syntax error. Alt 2.1: The system returns error message. Alt 2.2: The actor uses the correct locale data type or writes the correct syntax.

Table 4.3: *Use case description for insert data*

Use Case ID	UC-03
Use Case Name	Insert locale data
Actor	Database user
Purpose	Allow the database user to insert Amharic locale data in a table.
Pre-Condition	The actor must create the table with column type Ethiopian locale data types.
Flow of Events	1. The actor writes an insert value query and gives the Ethiopian locale data as input values. 2. Execute the written query [Alt 2].
Post-Condition	Conformation " value inserted"
Alternative Flow	[Alt 2]: if the input data is invalid or syntax error. Alt 2.1: the system returns error message. Alt 2.2: the actor insert the correct input or write the correct syntax.

Table 4.4: *Use case description for retrieve*

Use Case ID	UC-04
Use Case Name	View locale data.
Actor	Database user
Purpose	Allow the database user to view Amharic locale data stored in the database.
Pre-Condition	The actor must create the table.
Flow of Events	1. The actor writes a query to view the stored data and use the Ethiopian locale operator. 2. Execute the written query [Alt 2].
Post-Condition	value displayed to the user
Alternative Flow	[Alt 2]: If the operator is invalid or there is a syntax error. Alt 2.1: The system returns error message. Alt 2.2: The actor use the correct operator or write the correct syntax.

Table 4.5: *Use case description for modify recorded data*

Use Case ID	UC-05
Use Case Name	Modify locale record
Actor	Database user
Purpose	Allow the database user to modify the record of locale data stored in the table.
Pre-Condition	The actor must create the table and the target column must have a value.
Flow of Events	1. The actor writes an update query to modify the recorded value of the tables. 2. Execute the written query [Alt 2].
Post-Condition	Conformation " value updated"
Alternative Flow	[Alt 2]: If the table does not exist or there is a syntax error. Alt 2.1: The system returns error message. Alt 2.2: The actor writes the correct table name or writes the correct syntax.

Table 4.6: *Use case description of drop extension*

Use Case ID	UC-06
Use Case Name	Drop extension
Actor	Database user
Purpose	Allow the database user to drop(remove) Ethiopian locale extension from PostgreSQL.
Pre-Condition	The actor must add the extension.
Flow of Events	1. The actor writes drop extension query on the query tool. 2. Execute the written query [Alt 2].
Post-Condition	Conformation " extension dropped"
Alternative Flow	[Alt 2]: if the extension does not exist or syntax error. Alt 2.1: the system returns error message. Alt 2.2: the actor writes the correct extension name or writes the correct syntax.

4.3.2 Class Diagram

We use a class diagram to describes the structure of the proposed system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. The class diagram is the main building block of object-oriented modeling. It is used for general conceptual modeling of the structure of the application, and for detailed modeling translating the models into programming code. The classes in a class diagram represent structures and interactions of objects in the application. The extension consists of Table, EthiopianDate, EthiopianMoney, Miscellaneous classes. The class diagram of the Amharic locale extension module is shown in Figure 4.2.

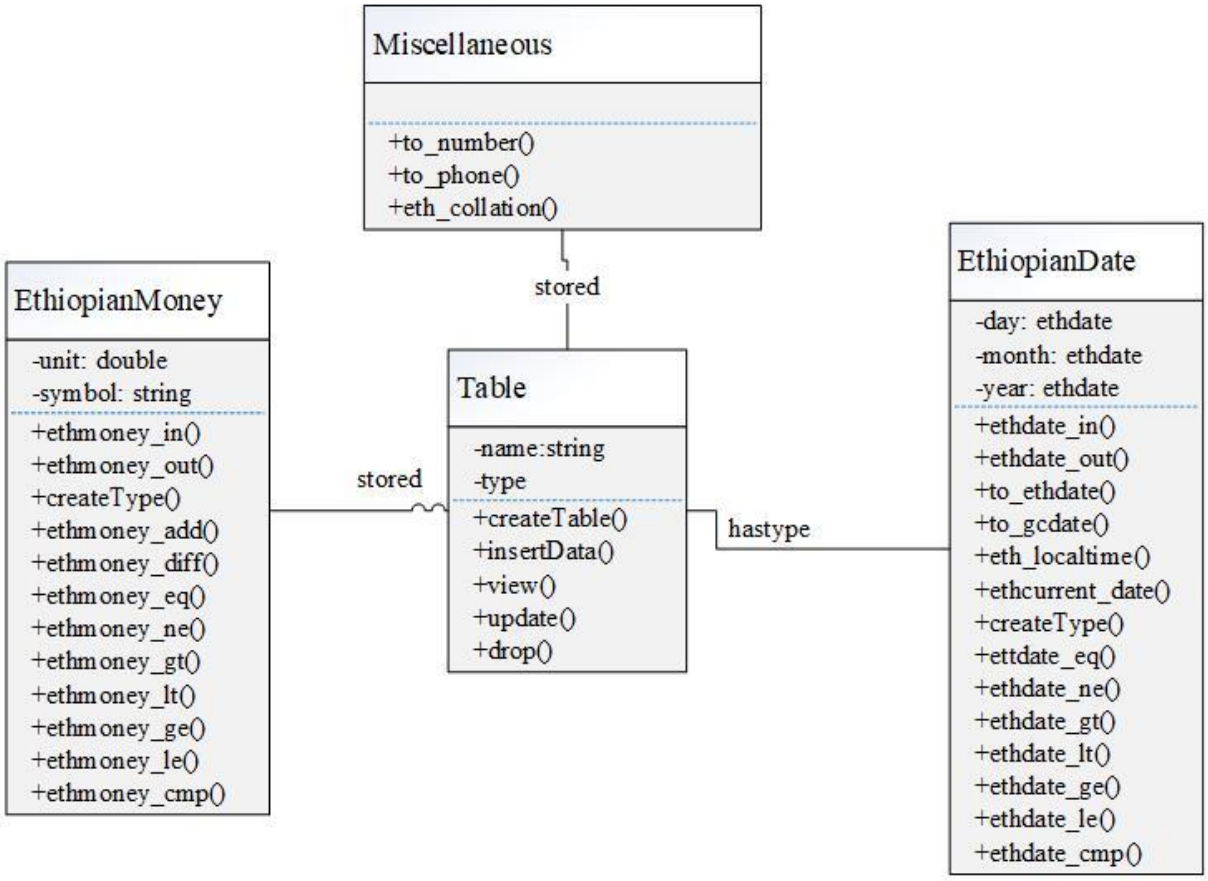

Figure 4.2: *Class diagram of Amharic locale extension*

4.3.3 Sequence Diagram

A sequence diagram is an interaction diagram that emphasizes the time ordering of messages. A sequence diagram shows a set of messages sent and received by the objects. Since it is useful for identifying additional objects that participate in the use cases and describe patterns

of communication among a set of interacting objects, the diagram is developed for each use case. An object interacts with another object by sending messages. The reception of a message by an object triggers the execution of an operation, which in turn may send messages to other objects. Arguments may be passed along with a message and are bound to the parameters of the executing operation in the receiving object. The sequence diagrams are included in this section and in Annex A.

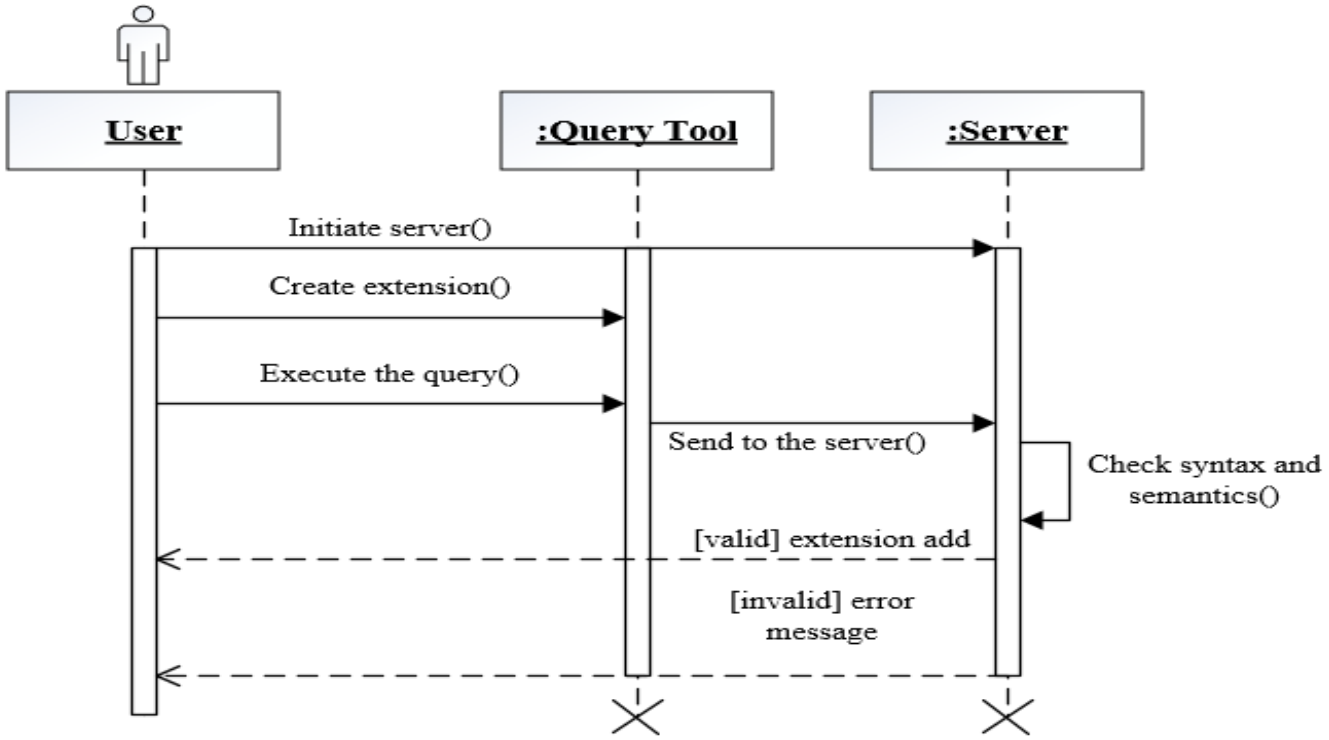

Figure 4.3: *Sequence diagram of load extension use case*

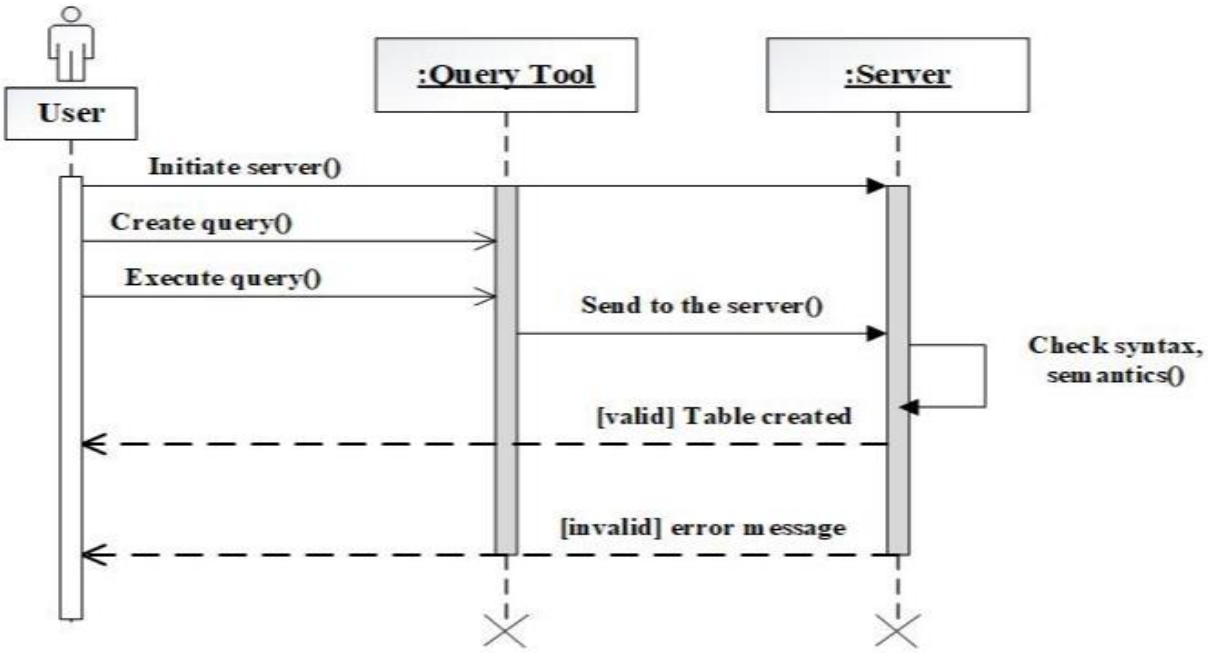

Figure 4.4: *Sequence diagram of create table use case*

Chapter Five – Integrating Amharic Locale in Database

During requirement gathering and analysis phase the services that are intended to be provided by the system are identified and modeled. At this phase, we specify the mechanism of how to organize the system internally by transforming of each functional requirement in the System analysis into system components. The main purpose of the design is to specify a solution domain for the analysis. This chapter presents design goals of the system, mapping principles and specifications and the architecture of the system.

5.1 Design Goals

Defining design goals is the first step of the system design, which describe the issues the proposed system should focus on. They arise from the nonfunctional requirements of the system described earlier.

End User Criteria

Usability: According to the ISO 9241-11:1998 standard "Usability is the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use". From the end users' perspective, the system should be designed in such a way that it is easy to learn and use by providing the extension using Ethiopic locale convention.

5.2 System Architecture

In this section, a detailed description of PostgreSQL database system architecture for adding extensions and the development of Amharic locale are presented. The chapter contains descriptions of the components comprising the architectural extensions, with detailed descriptions of the services provided by each component. For supporting of Amharic locale, we need to develop Amharic locale schema and incorporate it to the database by using the PostgreSQL database provided extension architecture. Amharic locale development includes multiple objects of the locale; for example, we propose a new data types associated with database components for making the database system to handle Amharic locale. These new data types will require new functions and new operators for managing and manipulating Amharic locale in a database. It is helpful to collect all these objects into a single package to

simplify database management. To define an extension, we need a resource file that contains the SQL commands to create the locale objects and a control file that specifies basic properties of the locale itself. The support of these aspects of locale is partitioned at both query language and architectural levels. Architectural support is provided for the addition and modification of the database management system components (especially in PostgreSQL) that impose a particular interpretation on locale values. As shown in Figure 5.1, the architecture to extend PostgreSQL DBMS is composed of two major components: PostgreSQL server and extension module. The PostgreSQL server component is responsible for performing query processing and transaction management. It is composed of two subcomponents: query processing and storage. The query-processing sub component must implement changes to a parser to take into account the SQL syntax and for converting input Unicode strings to Amharic locale data types. The storage sub-component manages the storage of the encoded data/ database and the database catalog. To support locale specification at all levels of a database (database catalog, schema, table, record and attribute) we proposed a new data types. The data types implemented as the storage format for Amharic locale data with rich semantics that capture the intuitive and familiar concepts of Amharic locale and associated with DBMS components for making the process of the database system can be fully supportd Amharic locale convention.

The Extension module (in our context the Amharic locale extension module) is added to the existing PostgreSQL database to manage Amharic locale data. This makes the database catalog be enhanced to model Amharic locale data types and different features of the locales (classes, attributes, functions, types, access methods, etc.) are tightly integrated in the schema to store and process this data by loading the module to PostgreSQL server. A simple create command modify many of these catalog. We propose wider adoption of locale convention by developing Amharic locale extension module. The availability of such module in Amharic locale will help to make implementation of locale operators possible.

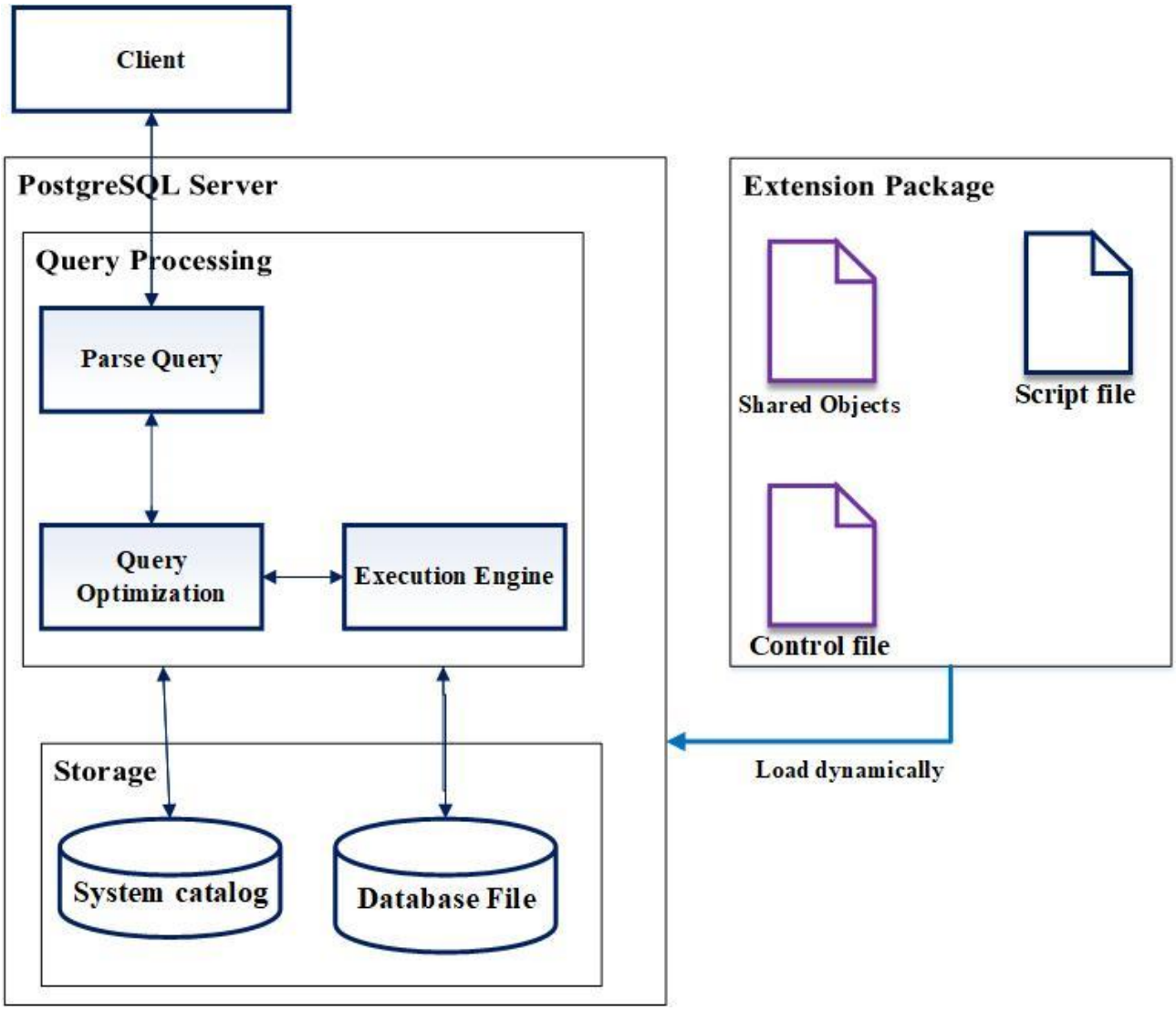

Figure 5.1: *PostgreSQL database system architecture for adding extension*

5.2.1 PostgreSQL Server

The PostgreSQL Server component is responsible for managing the database files, accepting connections to the database from client applications, and performing actions on the database on behalf of the clients. This component has two sub components.

Query Processing

Query processor takes the execution plan and interacts with the storage module to fetch the data from storage, and presents the data to the client. Query processing in locale data management environments could vary from being a simple string matching (in Ethiopic

scripts) to a complex semantic query. It requires locale convention query processing way. Since no Amharic locale conventions is specified in SQL standards, each DBMS has taken their own approach for handling such queries based on Romanization of the Amharic script. This component is extended to support the Amharic locale data. This component is responsible for invoking the Amharic locale resource file when the DBMS required for a certain operation performed on Amharic locale data. The extension module provides operations that support the query processing system at both compile-time and run-time. The query processing system invokes the Ethiopic locale file during query parsing to perform database tasks with Ethiopic cultural convention, type checking and binding of Amharic defined functions. The query-processing component consists a number of stages. These are query parser, query optimization and executor.

Query parser stage checks the query for correct syntax, semantics and creates a query tree. Then takes the query tree created and looks for any rules (stored in the system catalogs) to apply to the query tree. It performs the transformations given in the rule bodies.

The planner/optimizer takes the query tree and creates a query plan that will be the input to the executor. It does so by first creating all possible paths leading to the same result. Next the cost for the execution of each plan is estimated and the cheapest plan is chosen and handed back.

The executor recursively steps through the plan tree and retrieves rows in the way represented by the plan. The executor makes use of the storage system while scanning relations, performs sorts and joins, evaluates qualifications and finally hands back the rows derived.

Storage

The idea of every database is to store data such as images, texts, and even media files. Just like everything else, the same goes for data storing where it will require spaces for them to be stored. So, the basic Amharic locale support requirement is that the database system must be capable of storing data in Amharic locale conventions. While in specific instances it may be necessary to restrict the data stored in a specific convention, but we should not make such restriction universal. In this system architecture, the storage component is responsible to store for the encoded data. Since the Client component needs to be manipulated the data using the

locale convention encoding in order to handle the Amharic locale data, the same goes for the storage component, therefore the locale of the storage needs to be set to Ethiopic locale.

5.2.2 Amharic Extension Module

Amharic has its own locale convention which makes it differ from the other world. For example, it uses its own currency symbol, numerals, language, date and time system. These locales also need to manage according to Amharic locale convention in a database. To support such options, we add meta data of Amharic locale to the system catalog because when processing queries, the engine needs to utilize this information. We developed Amharic extension module that can be loaded into PostgreSQL server as needed. Once we load it into the database, this extension can function as built-in features. This extension is created in the form of a dynamically loadable object module. This extensions module consists of a dynamically loadable library (shared object), configuration (control) file and SQL files (script). Packaging all these files as an Amharic locale extension gives the benefit that PostgreSQL will recognize them as a single unit and it is helpful to simplify database management. There are two phases to add this extension module to the PostgreSQL server. First, we create the extension in C language and then compiling it into a dynamic object module (.dll or .so). Next, load the extension module to PostgreSQL server by *CREATE EXTENSION* command.

Shared Object

Shared object is a file containing compiled code from various object files that implement a new type or function stuffed into a single file. In our context, the shared object (*am_ET.so*) contains information about the new data types, functions and operators and PostgreSQL will load it as required. Generally, it contains information about Amharic locales. For example about the new datatypes (Ethdate, Ethmoney), functions (ethdate_in(), ethdate_out(), to_ethdate(), to_gcdate(), ethcurrent_date()) and operators of these new data types (+, -, <, <=, >, >=, =) for manipulating the data according to Amharic locale. The real implementation of these object files are developed in C programming language.

C Code

We identified locales which universally explain cultural-dependent aspects of a data. More specifically, there are properties that define the internal mechanisms of how data should be managed in a database. After the locales are identified we need a real implementation to support Amharic locale in database system. This C code is real implementation of Amharic locale extension. It includes C implementations of functions, types and operators.

As we mentioned earlier, to support Amharic locale in PostgreSQL we proposed new data types. New data type creation requires implementing external form and the internal form of a value. These functions (internal and external forms) determine how the type appears for input by the user, output to the user and how the type is organized in memory. The input function takes a null-terminated string external form of value as its argument and returns the memory representation of the type.

The output function takes the internal representation of the type as argument and returns a null-terminated character string. When we define a new data type, we need to tell PostgreSQL about external, internal form of the data type and how to convert a value from external form to internal form and from internal form to external form. We also define operators for performing operations with the data that belongs to the new data types. Such real implementations of the extension are compiled into dynamically loadable objects using *makefile* and is loaded to the server when needed. Sample source code of *makefile* is included in Annex D.

Listing 5.1: Algorithm for Ethiopian date internal form

```
if month name is equal with predefined Ethiopian month name
        the month is correct for Ethiopian calendar
if day is equal with predefined Ethiopian day values
       the month is correct for Ethiopian calendar
If Ethiopian month is between 1 and 12
         Ethiopian date is between 1 and 30
     Else
        If (Ethiopian month is ጳጉሜ or 13)
          If is Ethiopian leap year true
            Ethiopian date is between 1 and 6
```

```
        Else
            Ethiopian date is between 1 and 5.
if day, month and year are in allowed range
 store them as correct Ethiopian date value
```

Listing 5.2: Algorithm for Ethiopian date external form

```
get Ethiopian date
  display as DD-MM-YYYY format
```

where YYYY represents the calendar year, MM is the ordinal number of month within the calendar and DD is the ordinal number of day within the calendar.

Listing 5.3: Algorithm for Ethiopian currency internal form

```
if money value is with Arabic number and the currency unit is
null or ብር
    store as correct Ethiopian currency
else if money value is with Ge'ez number and the currency unit
is null or ብር
    change to equivalent Arabic number and store as Ethiopian
currency
```

Listing 5.4: Algorithm for Ethiopian currency output (external) form

```
get Ethiopian currency value
  display currency value with currency unit ብር
```

Control file

The control file specifies properties/metadata about the Amharic locale extension, which tells the basics about extension to PostgreSQL to register in its system catalog, and must be placed in the installation's SHAREDIR/extension directory. The file is *am_ET.control*, Parameters inside this file follows the same convention as *postgresql.conf* file (i.e. *parameter_name = parameter_value*). The *am_ET.control* file contain information about Ethiopian locale extension:

- Directory containing the SQL script file of Ethiopian locale.
- Default version Ethiopian locale extension.
- Comment about Ethiopian locale extension.
- Character set encoding used by the script file.

The script file

SQL script is mapping file, which we used to map all the SQL function with the corresponding C function of the extension (*am_ET*). It contains SQL commands about functions, types and operators of the Amharic locale. It also includes the required DDL and DML operations of extension. The file is *am_ET—1.0.sql*. The sample source code is included in Annex C.

5.3 Hardware/Software Mapping

PostgreSQL uses a client/server model [23]. As mentioned above, the Amharic extension module is intended to be added on PostgreSQL server that consists of new type and operator. Figure 5.2 depicts the deployment diagram of Amharic locale extension module. This extension module is installed on PostgreSQL server. This database (PostgreSQL) server stores the data and database catalog to be managed by the underlying database management system. At the front end, any user can write any database query through existing query tool, to use services provided by the extension for managing Amharic locale data. PostgreSQL provides graphically designed interface for writing database query and send these database queries to the database server. Then, the database server returns the result to the query tool which is accessible to the database user.

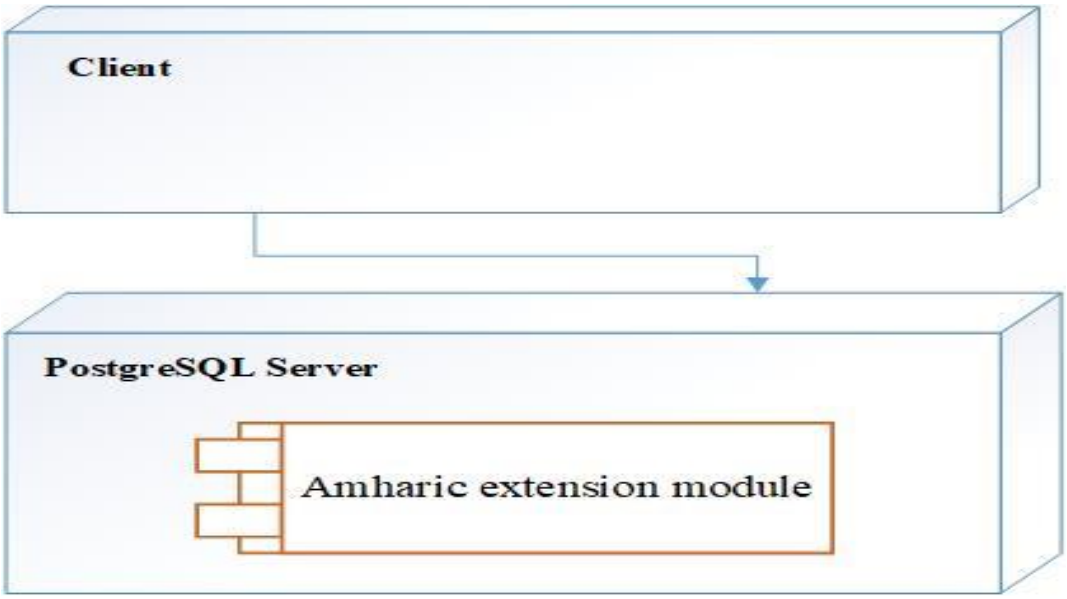

Figure 5.2: *Deployment diagram of Amharic locale extension*

Chapter Six – Implementation

This chapter provides the implementation details of Amharic locale integrating system in PostgreSQL database. The tools used in developing the extension and the developed extension are discussed in the following section.

6.1 Development Tools

To implement Amharic locale extension module which can meet the design goal, different tools and platforms have been used. These are:

- *C programming language:* is used to write the codes which will be executed by the server when a user wants to perform database operations.
- *PostgreSQL-12*: is used to produce header files and extension controls that needs to be included. To use a magic block we should include the header "*fmgr.h*", magic block allows the server to detect the compatibilities of the dynamically loaded object file. We always include "*postgres.h*" because it declares a number of things that we will need. For example:
    1. To specifies which C type corresponds to which SQL type when writing a C language function.
    2. To macro calling convention: overwhelm the complexity of passing arguments and results.
- *Ubuntu 18.10*: is used as an operating system.
- *Microsoft Office Visio 2016*: is used to design UML diagram.
- *Microsoft Office Word 2016*: is used for documentation.

6.2 Highlight about the Implementation

Load Extension: Once the code is written, it will be compiled and loaded to PostgreSQL database in order to use the functionality of Amharic locale extension. It can be done by writing and executing *CREATE EXTENSION am_ET* command on query tool after installing the extension module. This helps keeping track of all the extension objects together, create them at once to perform the other database operations as per what we need, and drop once with *DROP EXTENSION* command. Figure 6.2 shows loading the extension on PostgreSQL.

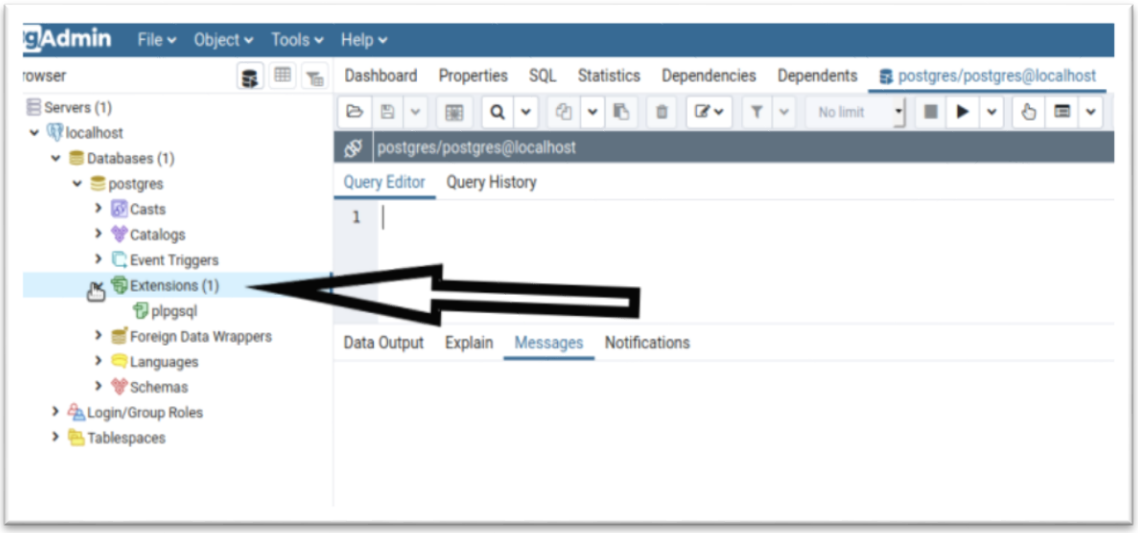

Figure 6.1: *Snapshot of PostgreSQL interface before loading Amharic locale extension*

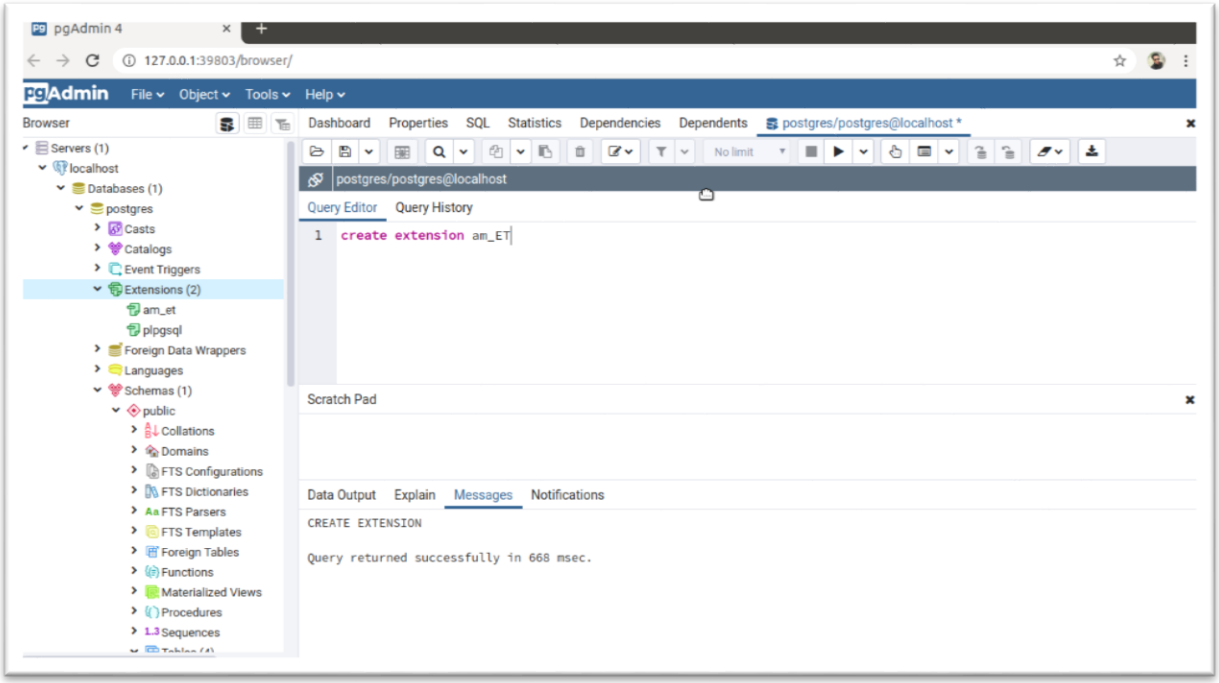

Figure 6.2: *Snapshot of loading Amharic locale extension*

Once the extension module is loaded into PostgreSQL server we can perform any database operations on Ethiopian locale data according to Amharic locale convention. In order to manage Amharic locale data, the database user must write SQL statement on a query tool and execute it. By using Amharic extension module, we can perform DDL and DML operations of a database. In the next sections we demonstrate the implementation of the extension module by selecting some of the functionalities.

Create table: In order to be able to add and manipulate data in database first we need to create a table. Each table contains one or more columns. Each column has an associated data type that defines the kind of data it can store. Figure 6.3 shows how the database user create a table with columns data type *Ethmoney* and *Ethdate*.

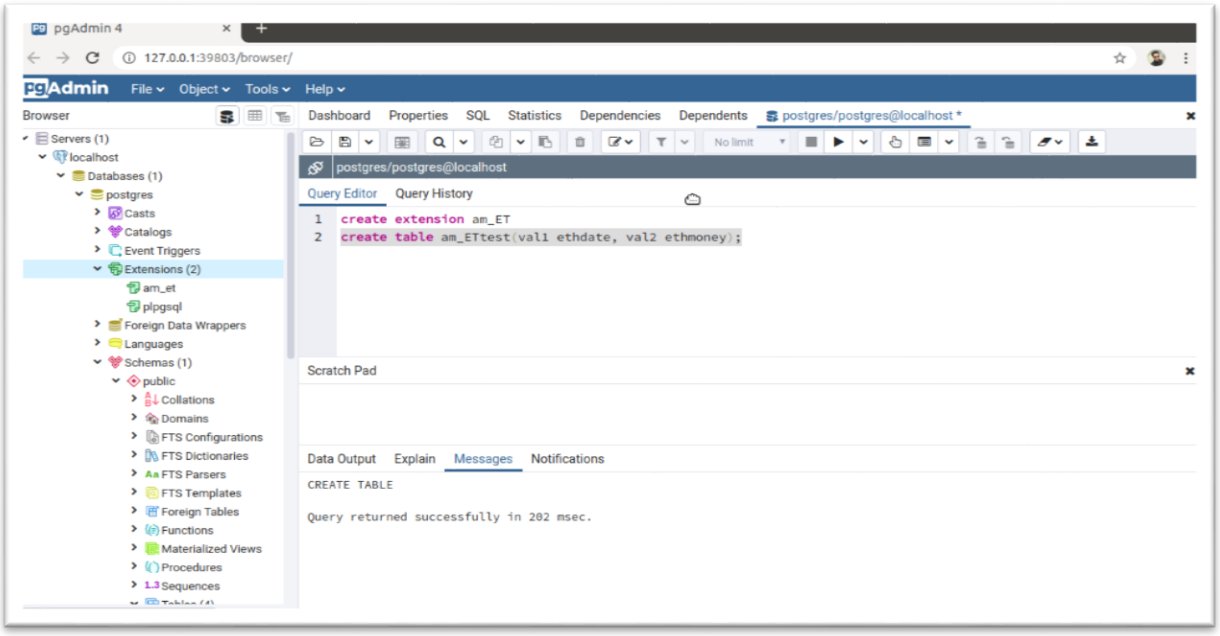

Figure 6.3: *Snapshot of create table query with Amharic locale column type*

Insert data: As shown in Figure 6.3, the database user can create a table to store Amharic locale data (money and date). To add values to the table we use *insert into* statement. Figure 6.4 shows how the database user add values for the columns of the table.

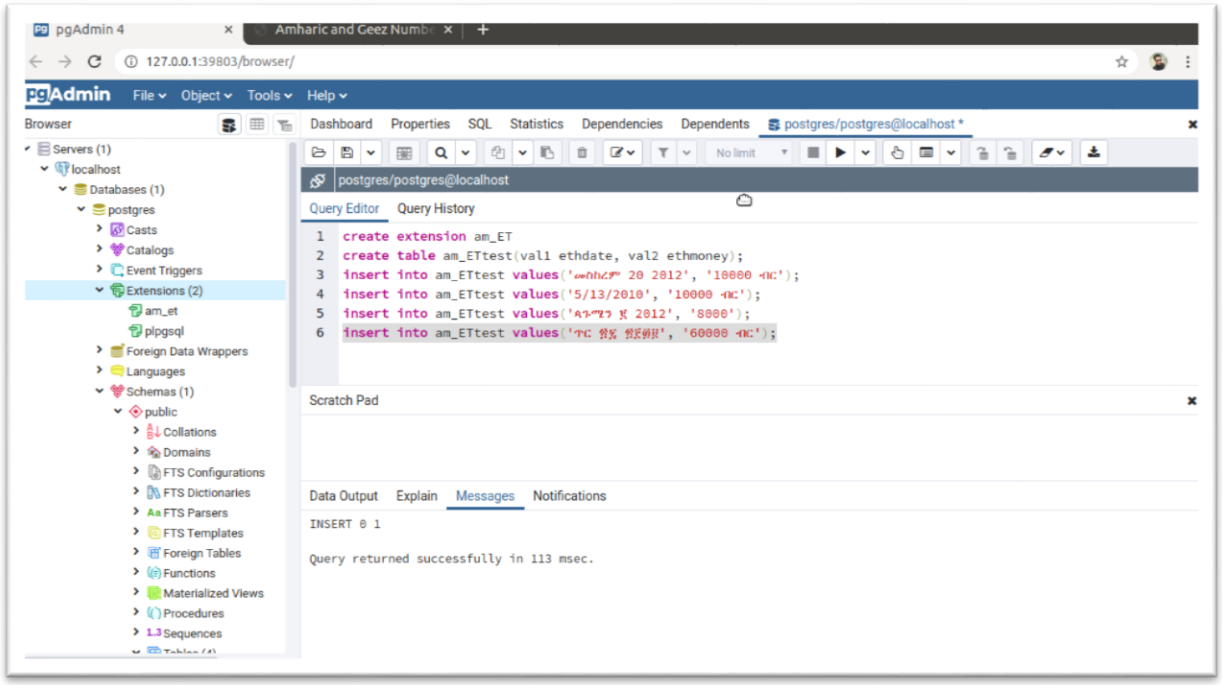

Figure 6.4: *Snapshot of insert into statement.*

Retrieve data and manipulate: To retrieve and manipulate the stored Amharic locale data in PostgreSQL we need to write select SQL statement with specified operator on the query tool and execute it. Figure 6.5 shows how the database user retrieve Amharic locale data ordered by date stored in a table.

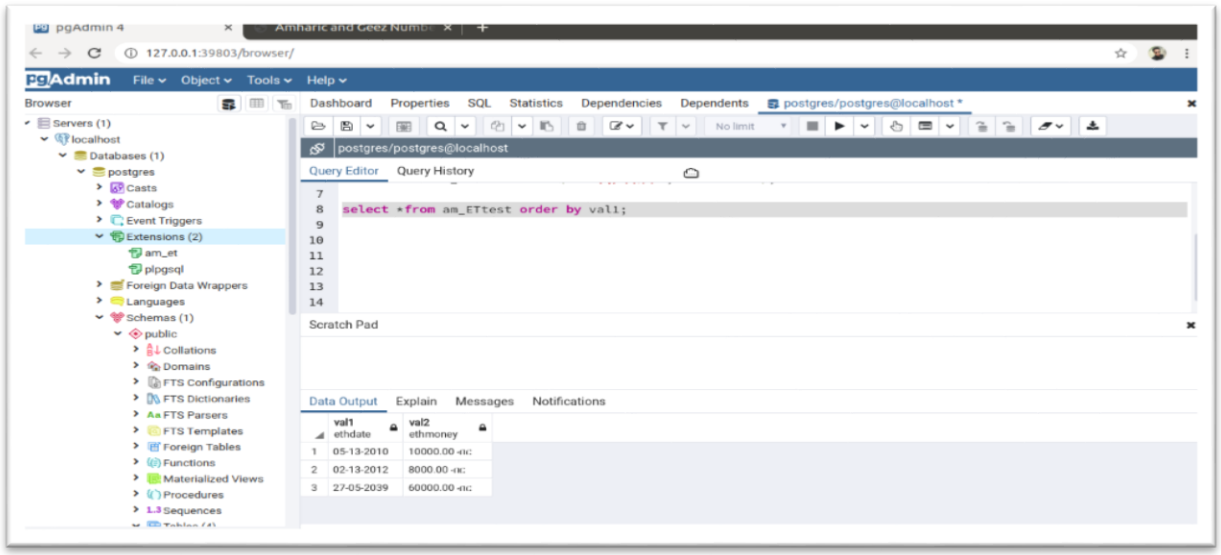

Figure 6.5: *Snapshot of select statement*

Manipulating the data: To manipulate Amharic locale data in PostgreSQL database use the predefined operators of Amharic locale extension (*am_ET*).

The other functionality of this extension module provided for a user is to allow users to convert Ethiopian date to Gregorian date and vice versa. To do this the user execute *to_gcdate* (*ethdate*) function and *to_ethdate* (date) respectively.

6.3 Testing Amharic Locale Extension Module

In order to make sure that the extension module is working correctly according to Amharic locale convention, unit and system testing were applied. The first testing is testing on the calendar. As stated in the previous part, the Ethiopian calendar is based on the Ethiopian Orthodox church computational practices. The church uses its own calculation to create a calendar for the country. The correctness of the calendar is tested by the calculation stated in ባህረ ሐሳብ. In addition to the calculation, the calendar is checked with other developed Ethiopian calendar system According to this test, this calendar is working correctly.

To the best of our knowledge, no other work deals with managing Amharic locale data according to Amharic locale convention in database system. As mentioned in Sections 3.1 and Section 3.3 other approaches do not consider Amharic locale convention logic. Therefore, experimental results to evaluate the proposed solution is presented in this section by comparing with the existing DBMS how different organization are managing Amharic locale data. In order to show how Amharic locale data handle in different organization by the existing DBMS and the created extension the following three tables were chosen: Employee, service and family tables.

EMPLOYEES TABLE

Field Name	Data Type
የማህደር ቁጥር	Short Text
አዲሱ የማህደር ቁጥር//	Short Text
የሰራተኛው ግብር ከፋይ መለያ ቁጥር	Short Text
የድርጅቱ የግብር ከፋይ መለያ ቁጥር/	Short Text
የአሰሪው መ/ቤት ሥም	Short Text
ተንቀሳቃሽ//**	Short Text
የአሰሪው መ/ቤት ስልክ ቁጥር//	Short Text
የአሰሪው መ/ቤት ፖ/ሳ/ቁ	Short Text
*የሰራተኛው ስም	Short Text
የአባት ስም///	Short Text
የእያት ስም//	Short Text
ፆታ	Short Text
የሰራተኛው የትውልድ ዘመን አ/ኢ/አ	Short Text
የእናት ስም/()	Short Text
/*/*የእናት አባት ስም	Short Text
/***የእናት አያት ስም	Short Text
የቅጥር ዘመን ///	Short Text
ጥቅል ደመወዝ/	Short Text
ዜግነት*ተገኘበት ሁኔታ****	Short Text
ዜግነት**//	Short Text
ክልል(መስተዳድር)	Short Text
ከፍለ ከተማ (ዞን)	Short Text
የከተማው ስም (ወረዳ))	Short Text
ቀበሌ/	Short Text
የቤት ቁጥር/)	Short Text
ስልክ ቁጥር(/)	Short Text
ፖ/ሳ/ቁ//	Short Text
ኢ/ሜ/ል/	Short Text
*የሚስት/የባል ስም	Short Text
*የሚስት/የባል አባት ስም	Short Text
*የሚስት/የባል እያት ስም/	Short Text
የሚስት/የባል የትውልድ ዘመን*	Short Text
*የሚስት/የባል ግብር ከፋይ መለያ ቁጥ	Short Text
/የተመዘገበበት ቀን/*	Short Text

Figure 6.6: *Screen shot of Employee table design of POESSA*

FAMILY TABLE

Field Name	Data Type
የማህደር ቁጥር	Short Text
አዲሱ የማህደርቁጥር*****	Short Text
የአባት ስም/	Short Text
የአባት አባት ስም(	Short Text
የአባት እያት ስም	Short Text
የአባት/ የትውልድ ዘመን*	Short Text
የአባት ወርሃዊ ገቢ/	Short Text
የአባት መተዳደሪያ ሁኔታ//	Short Text
የእናት ስም	Short Text
የእናት አባት ስም/)	Short Text
የእናት እያት ስም(/	Short Text
የእናት የትውልድ ዘመን(()	Short Text
የእናት ወርሃዊ ገቢ)))	Short Text
የእናት መተዳደሪያ ሁኔታ//))	Short Text
ወላጆች የሚያገፉበት ሁኔታ በገንዘብ ከሆነ ይገለፅ/	Short Text
ወላጆች የሚያገፉበት ሁኔታ ሌላ ይገለፅ	Short Text
የተመዘገበበት ቀን/ *	Short Text

Figure 6.7: *Screen shot of Family table design of POESSA*

SERVICE TABLE

Field Name	Data Type
የማህደር ቁጥር	Short Text
አዲሱ የማህደር ቁጥር//*/	Short Text
የግብር ከፋይ መለያ ቁጥር	Short Text
የድርጅቱ ስም ////***	Short Text
የአገልግሎት አይነት//	Short Text
የአገልግሎት መነሻ ቀን/ወር/ዓመት	Short Text
የአገልግሎት መድረሻ ቀን/ወር/ዓመት//	Short Text
ወርሃዊ ደመወዝ	Money
ድርጅቱ ፕሮቪደንት ገቢ አድርጓል ከመቼ ጀምሮ	Short Text
ቅጥር ያበቃበት ምክንያት/	Long Text
ሲሰናበቱ ይከፈል የነበረ ደመወዝ	Money
/የመዝጋቢው ሠራተኛ ስም/	Short Text
የተመዘገበበት ቀን/*	Short Text

Figure 6.8*: Screen shot of Service table design of POESSA*

The tables, do have a minimum of 19 columns in total to store Amharic locale values. As shown in Figure 6.12 and Figure 6.13 to store Amharic locales (date and time, currency, phone number and numerals) in the current database system for column like የትውልድ ዘመን አ/ኢ/አ (birth date), ወርሃዊ ደመወዝ (salary), የተቀጠረበት ቀን (date of hire), የተመዘገበበት ቀን (registration date), ስልክ ቁጥር (phone number) and others the database users use text data type. Figure 6.14 shows that the database users use money data type for a column ወርሃዊ ደመወዝ (salary) and text data type for columns related with date and time.

In order to compare how the Amharic locale data is managed by using Amharic locale extension we presented the table design of Employee, Service and family tables of POESSA for calendar and currency locales. As shown below in the Figure 6.15, Figure 6.16 and Figure 6.17 we are using a new defined data type *ethdate* and *ethmoney* for storing currency, date and time values in PostgreSQL database respectively as described in this document. Since the required operations related with Ethiopic numerals, Ethiopian currency and Ethiopian calendar are defined in the extension we can perform operation as we want.

To store phone number by using this extension we can use varchar /text /char data type with fixed length and use function *to_phone*() to clear the difference between phone number with other text value.

To manage geez number by using this extension we can use integer data type and use function *to_number*() because the existing systems assume geez numbers as glyph character.

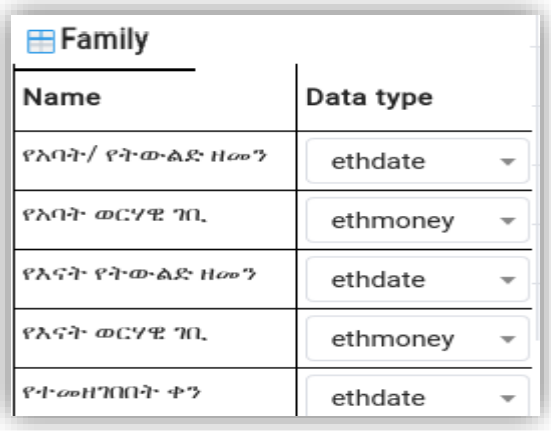

Figure 6.9: *Screen shot of Family table design after installing am_ET*

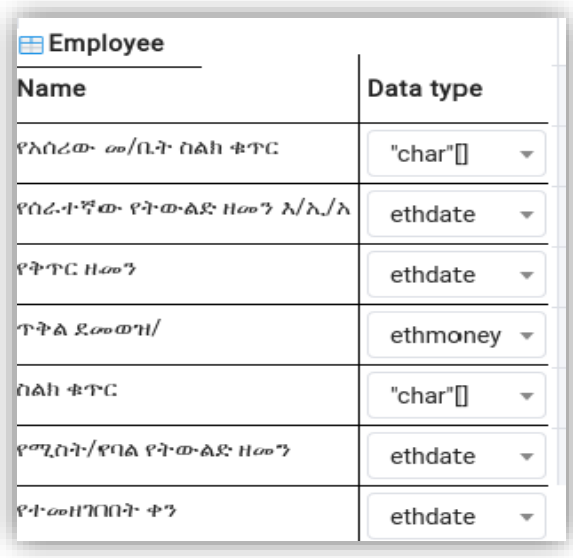

Figure 6.10: *Screen shot of Employee table design after installing am_ET*

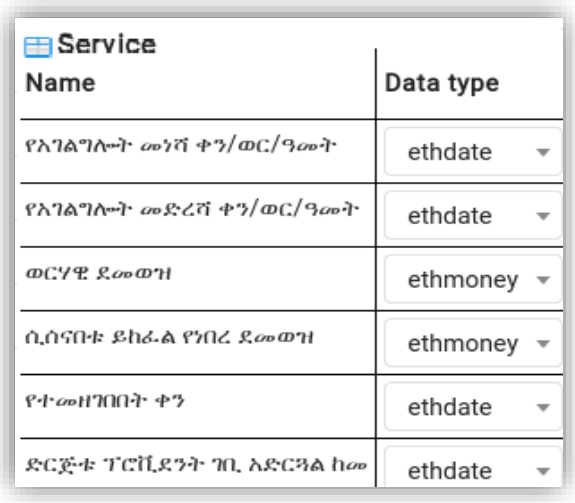

Figure 6.11: *Screen shot of Service table design after installing am_ET*

After making the comparison the following conclusions were made:

- According to Figure 6.14 managing Amharic locale data in the existing DBMS with western locale convention leads to miss information. For example, when inserting a value for column ወርሃዊ ደመወዝ (salary) the amount is stored in $.
- In the existing DBMS we can't perform the required operations to manipulate Amharic locale data. However, by using the developed extension we can do different operations on Amharic locale data according to the locale convention.
- Even though Amharic locale data is managed by the existing DBMS, according to Figure 6.12 and Figure 6.13 the data management way is noticeably exposed to error. i.e. we can insert any invalid value for the column salary, date of birth, date of hire, Phone number and registration date. However, these kinds of errors are handle by appropriate exception handling mechanisms on the developed extension.

Usability Testing

We conducted user acceptance testing to evaluate Amharic locale extension functionalities which was performed based on the ISO 9241-11 usability testing attributes, such as efficiency, effectiveness (refers to completeness and accuracy with which users achieve certain goals) and user satisfaction.

In the evaluation, 2 different types of user's categories are involved. 5 database administrators and 20 database users were participated in the testing. Before conducting the evaluation process, detailed description about the extension has been given to the participants as it helps them in having an insight to the developed extension. After the demonstration of the developed extension, participants were provided with respective questionnaires. The questionnaires were used to capture their sense of satisfaction concerning the developed extension. The designed questionnaire can be found in Annex B. A five level likert scale (strongly agree (5), agree (4), less agree (3), disagree (2) and strongly disagree (1)) is used for the responses of the questions because it is easy to interpret.

After getting response from respondents through the questionnaire, we calculated each scale of questionnaire. Table 6.1 summarizes the respondents' result of the extension questionnaire of each question in number for the two categories of users. According to the respondent, the Amharic locale extension is suitable to manage Amharic locale data. As the result shows in Table 6.1, the questionnaire which are related to storing, querying and presenting Amharic locale data in PostgreSQL showed that most of the respondents (database administrators and database users) agree and strongly agree; very few of respondents responds less agree, disagree and strongly disagree.

As shown in Table 6.2 we analyze the two categories of the users from the questionnaire results that the percentage of strongly agree for database administrators and database users was 60% and 49.3% respectively. 25.7% database administrators and 44.3% users were responds agree. 8.6% of the administrator and 6.4% of users were less agree. Similarly, 5.7% administrators and 0% of users were disagree. Both administrators and users were 0% strongly disagree.

Table 6.1: *Detailed summary of questionnaire result*

Question No.	**Admin**					**User**				
	SA	**Ag**	**LA**	**Di**	**SD**	**SA**	**Ag**	**LA**	**Di**	**SD**
1	4	1	0	0	0	13	7	0	0	0
2	5	0	0	0	0	9	10	1	0	0
3	3	2	0	0	0	10	10	0	0	0
4	3	2	0	0	0	11	7	2	0	0
5	1	1	2	1	0	7	10	3	0	0
6	4	1	0	0	0	9	10	1	0	0
7	1	2	1	1	0	10	8	2	0	0

Key: SA= Strongly Agree, Ag=Agree, LA=Less Agree, Di= Disagree, SD= Strongly Disagree

Table 6.2: *Analysis of degree usability testing for the two users categories*

Answer	**Administrator**		**User**	
	N	**%**	**N**	**%**
Strongly Agree	21	60	69	49.3
Agree	9	25.7	62	44.3
Less Agree	3	8.6	9	6.4
Disagree	2	5.7	0	0
Strongly Disagree	0	0	0	0
	35	100	140	100

N= Number of occurrences for each degree on scale marked by users.

%= Percentages of each question answered

Chapter Seven – Conclusion and Future Work

7.1 Conclusion

Due to the continuous rise of local data in many domains, the question of how to manage these data efficiently based on the locale requirement of the user has become of crucial importance. However, both internationally and nationally, most of the data management software are developed to support English language. Apart from government efforts, individual developers also need to be motivated to take part in localized data management development process.

In this project we have made the first step towards the ultimate objective of achieving complete multi-locale functionality in database systems. We first highlighted the need of managing locale data according to locale convention, with motivating example from real-life domain. We presented Amharic locales and their properties that made them differ from other locales. Then, we proposed an extension to database management tool to address the problem of Ethiopian locale data management in the existing database management system.

Considering the properties of Amharic locale, we have developed Amharic locale extension to manage Ethiopian locale data in PostgreSQL database, which can handle currency, numerals, calendar, date and time according Amharic locale logic/ convention. Lastly the implemented extension was tested and the result shows that it works correctly.

7.2 Contribution

The contributions of this project work are summarized as follows:

- Designed internal and external forms of Ethiopian calendar to handle by PostgreSQL database.
- Extending PostgreSQL database to support Ethiopian date and time.
- Designed algorithms to convert Ethiopic calendar to European or Gregorian Calendar and vice versa.
- Designed algorithms to convert Ethiopic (Ge’ez) number to Arabic number.
- Developed an extension to support Ethiopian phone numbers in PostgreSQL database.
- Designed and implemented database extension for PostgreSQL that support Ethiopic script numerals.

- Designed and implemented an extension for PostgreSQL database that support Ethiopic script day, month and year entry.
- Demonstrated the usability of developed extension on the existing PostgreSQL database.

7.3 Future work

Even though the implemented extension does work correctly, considerable amount of additional development has to be done to enhance the usability of the extension.

- Implementing new database indexing for Ethiopic script.
- Using this extension in at operating system level.

References

[1] M. Huijs, S. Jansen and S. Brinkkemper, "Internationalization and Export of Software Products," in *International Conference of Software Business*, Switzerland, 2015.

[2] S. Moura, "Databases Internationalization Model," *12th Iberian Conference on Information Systems and Technologies*, 2017.

[3] C. Lako, "on internationalization (I18N)," 2015.

[4] X. Xia, D. Lo, F. Zhu, X. Wang, and B. Zhou, "Software Internationalization and Localization: An Industrial Experience," in *International Conference on Engineering of Complex Computer Systems*, 2013.

[5] M. Bhatia and A. Sharma, "A Survey of Software Localization Work," *Journal of Global Research in Computer Science*, vol. 4, 2013.

[6] V. Dagient and R. Laucius, "Internationalization of Open Source Software: Framework and Some Issues," in *second International Conference Information Technology: Research and Education*, London, England, 2004.

[7] Unicode Consortium, "The Unicode Standard," [Online]. Available: http://www.unicode.org/versions/Unicode10.0.0/UnicodeStandard-10.0.pdf. [Accessed 12 december 2019].

[8] Million Meshesha and C.V. Jawahar, "Indigenous scripts of African languages," *Indilinga African Journal of Indigenous Knowledge Systems*, vol. 6, 2007.

[9] B. Beyene, O. Kummer and M. Kudlek , "Ethiopian Language Support for the Babel Package," University of Hamburg, Department of Computer Science, 2006.

[10] Zenebe Nigussie, "Perpetual Ethiopic and European Calendar and Organizer System for an Android Based Smart Phone," Unpublished Master Thesis, Department of Computer Science, Addis Ababa University, 2010.

[11] Getnet Mossie and Metages Molla, "The impact of using Gregorian calendar dates in systems that adapt localization: In the case of Ethiopia," *IOSR Journal of Computer Engineering,* vol. Volume 19, 2017.

[12] Daniel Yacob, "Localize or be Localized," *International Symposium on ICT Education and Application in Developing Countries,* 2004.

[13] W. Ding, P. Liang, A. Tang, H. v. Vliet and M. Shahin, "How Do Open Source Communities Document Software Architecture: An Exploratory Survey," *19th International Conference on Engineering of Complex Computer Systems,* 2014.

[14] V. R. Sánchez, P. N. Ayuso, J. A. Galindo and D. Benavides, "Open Source Adoption Factors - A Systematic Literature Review," in *10th Annual Future of Open Source Survey*, 2020.

[15] M. Silić, "Enterprise Open Source Software Adoption Dilemma: Influence of the Information Technology Risk Factors," *Unpublished Dissertation, University of St. Gallen,* 2015.

[16] M. Heron, V. Hanson and I. Ricketts, "Open source and accessibility: advantages and limitations," *Journal of Interaction Science,* 2013.

[17] G. Sood, Shipra and R. Soni, "Comparative Study: Proprietary Software vs. Open Source Software," *International Journal of Innovative Research in Computer and Communication Engineering,* vol. 4, 2016.

[18] F. Kon, P. Meirelles, N. Lago, A. Terceiro, C. Chavez and M. Mendonça, "Free and Open Source Software Development and Research: Opportunities for Software Engineering," *25th Brazilian Symposium on Software Engineering Research: Opportunities for Software Engineering,* 2011.

[19] A. Singh, R.K Bansal and N. Jha, "Open Source Software vs Proprietary Software," *International Journal of Computer Applications,* vol. 114, 2015.

[20] R. Elmasri, Fundamentals of Database Systems, six edition, 2011.

[21] V. Dagiene, "Research on Open Source Software Intended to Promote its Usage in Education," *19th IFIP World Computer Congress,* 2006.

[22] DB-Engines, "Popularity of open source DBMS versus commercial DBMS," [Online]. Available: https://db-engines.com/en/ranking_osvsc. [Accessed 6 june 2020].

[23] G. Sakarkar, V . M. Thakar, "Autonomous Software Agent for Localization," *International Conference on Intelligent Agent & Multi-Agent Systems,* 2009.

[24] M. Davis, "Unicode Local Data Markup Language," [Online]. Available: http://www.unicode.org/reports/tr35/. [Accessed 13 December 2019].

[25] W. Bader, "Website Localization Techniques," *International Journal of Computer Applications,* vol. 150, 2016.

[26] Lingobit Technologies, "Database Localization," [Online]. Available: http://www.lingobit.com/solutions/database/database_localization.html. [Accessed 28 december 2019].

[27] Rufael Tadesse and Fekade Getahun, "Localization of Open Source Web Content Management System," *HiLCoE Journal of Computer Science and Technology,* vol. 2, 2014.

[28] R. W. Collins, "Software Localization: Issues and Methods," *The 9th European Conference on Information Systems,* 2001.

[29] R. Lammel and Ostermann, "Software Extension and Integration with Type Classes," p. 22–26, 2006.

[30] PostgreSQL Global Development Group, "PostgreSQL Documentation," University of California, [Online]. Available: https://www.postgresql.org/docs/10/static/index.html. [Accessed 21 January 2020].

[31] H. Krosing, J. Mlodgenski and U. Dar,, "PostgreSQL Server Programming, Second edition," Packt Publishing Ltd, 2015.

[32] R. Bhatiya, "Oracle Database Globalization Support Guide, 12c Release 1," Oracle, [Online]. Available: https://docs.oracle.com/cd/E11882_01/server.112/e10729.pdf. [Accessed 21 December 2019].

[33] D. Axmark and M. Widenius, "Reference Manual for the MySQL Database System," Oracle, [Online]. Available: https://downloads.mysql.com/docs/refman-8.0-en.pdf. [Accessed 21 december 2019].

[34] Clear ICT Solutions, "Ethiopian Localization Project," [Online]. Available: https://github.com/OCA/l10n-ethiopia. [Accessed 21 june 2020].

Annex A: Sequence Diagrams of Some Use Cases in the extension

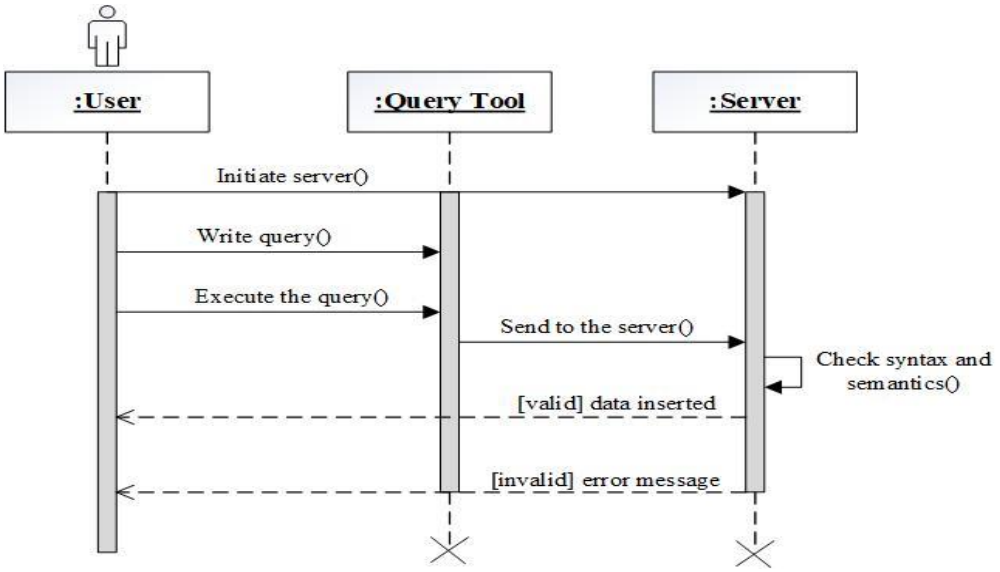

Figure A.1: *Sequence diagram of insert data use case*

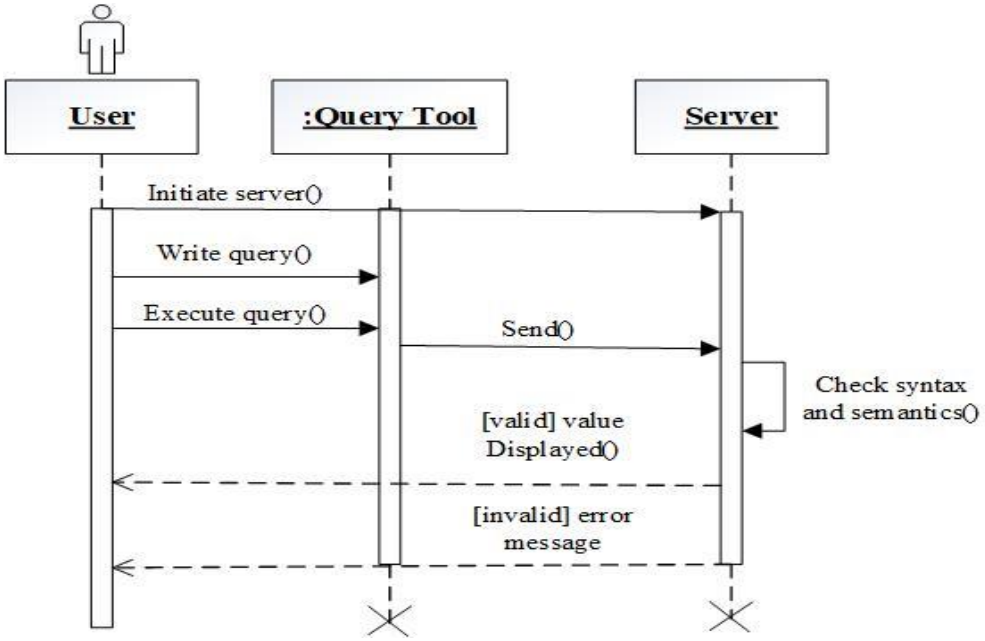

Figure A.2: *Sequence diagram of view data use case*

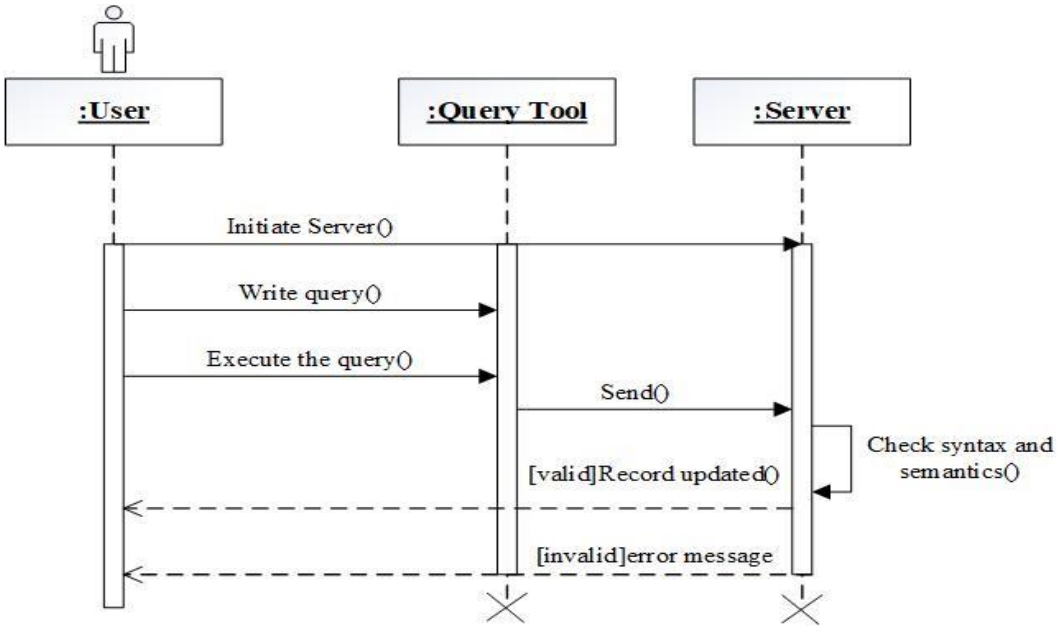

Figure A.3: *Sequence diagram of modify use case*

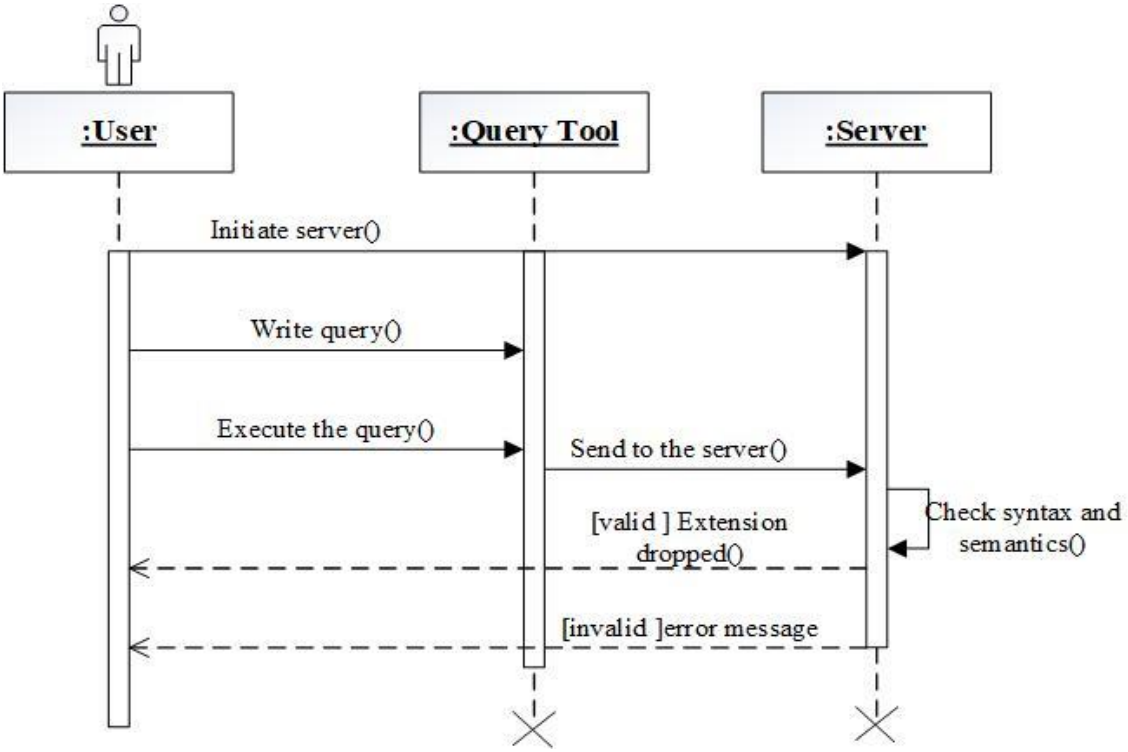

Figure A.4: *Sequence diagram of drop extension use case*

Annex B: Usability Testing Questionnaire

This questionnaire is intended to know user satisfaction related to the functionality of the developed extension. Please indicate your agreement by making ✓ in the boxes.

No	Questionnaire for Usability Evaluation	Score				
		1	2	3	4	5
1.	The extension is easy to use.					
2.	In relation to the existing DBMS, the extension is more appropriate to manage Amharic locale data according to Amharic locale convention.					
3.	The extension minimizes occurrence of errors when managing Amharic locale data and give enough description when the error occurs?					
4.	The extension is good enough to manage (store, retrieve and present) Amharic locale data (calendar, geez numerals, currency and phone number).					
5.	All the expected function of the Amharic locale was present.					
6.	The extension can be used by any database user with basic knowledge of using the existing DBMS?					
7.	The extension matches your expectation to manage (store, query and present) Amharic locale (calendar, currency, numerals phone number etc..).					

Key: 1= Strongly agree; 2= Agree; 3= Less Agree; 4= Disagree; 5= Strongly Disagree

Please write any other comments about the Ethiopian locale extension

_____

_____

_____

Annex C: Script file Source Code

Sample source code of Amharic locale extension script file am_ET--1.0.sql.

```
/* contrib/am_ET/am_ET--1.0.sql */
-- complain if script is sourced in psql, rather than via
CREATE EXTENSION
\echo Use "CREATE EXTENSION am_ET" to load this file. \quit
CREATE TYPE ethmoney;
CREATE OR REPLACE FUNCTION
ethmoney_in(cstring)RETURNS ethmoney AS 'MODULE_PATHNAME'
LANGUAGE C STRICT IMMUTABLE PARALLEL SAFE;
CREATE OR REPLACE FUNCTION ethmoney_out(ethmoney) RETURNS
cstring AS 'MODULE_PATHNAME' LANGUAGE C STRICT IMMUTABLE
PARALLEL SAFE;
CREATE FUNCTION ethmoney_recv(internal) RETURNS ethmoney
AS 'MODULE_PATHNAME' LANGUAGE C STRICT IMMUTABLE PARALLEL
SAFE;
CREATE FUNCTION ethmoney_send(ethmoney) RETURNS bytea AS
'MODULE_PATHNAME' LANGUAGE C STRICT IMMUTABLE PARALLEL
SAFE;
CREATE OR REPLACE FUNCTION
ethmoney_add(ethmoney, ethmoney) RETURNS ethmoney AS
'MODULE_PATHNAME','ethmoney_add'
LANGUAGE C STRICT;
CREATE OPERATOR + (
leftarg = ethmoney,
rightarg = ethmoney,
function = ethmoney_add,
commutator = +
);
```

Annex D: Source code of Makefile

Sample source code of Amharic locale extension *Makefile*.

```
MODULES = am_ET
EXTENSION = am_ET
DATA = am_ET--1.0.sql
PGFILEDESC = "am_ET - manage Amharic locale data in a database"
REGRESS = am_ET
PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)
include $(PGXS)
subdir = contrib/am_ET
```

Declaration

I, the undersigned, declare that this project is my original work and has not been presented for a degree in any other university, and that all source of materials used for the project have been duly acknowledged.

Declared by:

Name: **Yonathan Misgan**

Signature: _____

Date: June 29, 2020

Confirmed by advisor:

Name: **Dr. Solomon Atnafu**

Signature: _____

Date: June 29, 2020

Place and date of submission: Addis Ababa University, June, 2020.