GSoC 2023 PostgreSQL Project Proposal

Postgres extension tutorial

Basic Information:

- ***Name:*** Ishaan Adarsh
- ***GitHub:*** https://github.com/IshaanAdarsh
- ***Email:*** ishaanad9@gamil.com
- ***LinkedIn:*** https://www.linkedin.com/in/ishaan-adarsh-161a56222/
- ***Location:*** Bengaluru, India (GMT+5:30)

About me:

My name is Ishaan Adarsh, and I am a 2nd year Engineering Student at National Institute of Technology, Raipur. I am writing to express my interest in the Postgres extension tutorial / quick start (2023). I have experience in several programming languages, including C++, Solidity, HTML, CSS, ReactJS and Svelte Kit. In addition, I have experience with SQL and database design, which I believe will be valuable in developing an effective tutorial for the Postgres extension system.

Why I am interested in PostgreSQL :

I have always been passionate about open-source. However, I didn't have a chance to contribute to an actual open-source project. I hope to take the opportunity of GSoC to become a Postgres contributor and I think it's a good chance for me to get involved in the open-source community.

I believe that open-source software is an essential tool for promoting *collaboration* and *innovation*, and I am excited to have the opportunity to contribute to a project that will make it easier for developers to contribute to Postgres.

In addition to my technical skills, I am a strong communicator and collaborator. I believe that effective communication and collaboration are essential for success in any open-source project, and I am committed to working closely with the *Postgres community* to develop a tutorial that meets the needs of developers. Overall, I believe that my technical skills, passion for open-source software, and strong communication and collaboration skills make me an ideal candidate for the *Postgres Extension Tutorial and Quick Start Guide project.*

Deliverables:

1. A *comprehensive tutorial/quick start guide* for writing Postgres extensions, covering the following topics:
    - Prerequisites for writing an extension
    - Starting an extension
    - Writing a Makefile
    - Using PGXS and PGXN
    - Using Procedural Languages and External Languages
    - Writing regression tests
    - Release management and upgradability
    - The tutorial should be easy to follow, with *clear examples* and *step-by-step instructions*.

2. A sample Postgres extension:
    - To illustrate the concepts covered in the tutorial, a *sample Postgres extension* should be developed using *Python* and *JavaScript*.
    - The extension should be *well-documented* and include *regression tests* to ensure it is working correctly.

3. Documentation for the sample extension:
    - Documentation should be provided for the sample extension, including a README file and any necessary installation instructions.

4. Integration with the Postgres documentation system:
    - The tutorial and sample extension should be *integrated* into the official Postgres documentation system to ensure they are easily accessible to users.
    - A set of *best practices* and guidelines for extension development, covering topics such as *security*, *performance*, and *maintainability*.

5. Testing and validation:
    - The tutorial, sample extension, and documentation should be tested and *validated* to ensure they are working correctly and meet the requirements outlined in the project description.
    - *Collaboration* with other *Postgres developers* to incorporate feedback and suggestions for *improving* the extension development process.

6. Ongoing maintenance and support (Post Work Program):
    - The project should include *ongoing maintenance* and support for the tutorial, sample extension, and documentation to ensure they stay up-to-date with changes to Postgres and continue to meet the needs of the community.

Detailed Description:

The Postgres database management system is a *powerful* and *flexible platform* that allows developers to build complex applications. One of the key features of Postgres is its extension system, which allows developers to extend the functionality of the database by writing their own custom extensions. However, the process of writing a Postgres extension can be daunting for new developers, as the documentation is often *difficult* to navigate and the *learning curve is steep.*

To address this issue, we propose to create a *comprehensive tutorial* and quick start guide for writing Postgres extensions, with the aim of lowering the barrier to entry for new contributors and making it easier for experienced developers to create high-quality extensions.

It will cover the following topics:

1. Prerequisites for writing an extension:

This section will cover the knowledge of needed to write an extension

- Postgres
- Target languages (*C, Python, JavaScript*)
- Tools and Software's needed:
  a) PostgreSQL Server
  b) PGXS
  c) PGXN
  d) Git: Used in Postgres Extension Development to *manage* code changes, collaborate with other developers and maintain a history of the codebase.

To make this part of the tutorial more accessible to novice developers, we can provide:

- Step-by-step Instructions to install and configure each of these tools and software. We can also provide code examples and snippets to help developers get started quickly.
- *Provide links* to *additional resources* such as documentation and tutorials to help developers deepen their understanding of these tools and software.

2. Starting an extension:

This section will explain how to create a new extension, discuss the structure of an extension, and explain how to write extension code.

Problem:

Novice developers may not be familiar with the process of creating a new extension in Postgres or understanding the structure of an extension.

Solution:

- This can include the use of tools such as the `pgxn` command-line tool to automate the creation of extension scaffolding.
- We can provide explanations of the various components of an extension, such as the *control file and SQL script files*, to help developers understand the structure of an extension.
- To guide developers through writing extension code, we can provide examples of basic extension functionality, such as adding a new SQL function, with explanations of the code and how it works.
- We can provide information on debugging and troubleshooting extension code, such as using `pg_config` to check for *missing dependencies*.

3. **Writing a Makefile:** This section will discuss the role of a Makefile in extension development and provide an example Makefile.

Problem:
Novice developers may not be familiar with Makefiles and their role in extension development, which can make it difficult for them to build and manage their extensions.

Solution:

- Explain the *purpose* of Makefiles in extension development
- Create a sample Makefile that developers can use as a template for their own extensions. It should also include variables for defining the *extension name*, *version number*, and other metadata. The Makefile should include commands for
    - building the extension
    - running *regression tests*
    - installing the extension.

- Offer *troubleshooting advice* for common issues that may arise when building and installing extensions using Makefiles. This can include tips for debugging Makefile syntax errors or resolving dependency issues.

4. **Using PGXS and PGXN:** These sections will explain what PGXS and PGXN are, their role in extension development, and how to use them.

Problem:
Novice developers might not be familiar with PGXS and PGXN, which can make it difficult for them to build, install, and distribute their extensions.

Solution:

- The tutorial can start by *defining* what PGXS and PGXN are and why they are important in extension development.
- We can then provide a step-by-step guide on how to use PGXS to build and install an extension, along with code snippets and examples. We can also explain how to use PGXN to *distribute* the extension and make it available to other users.
- We can provide information on how to *test* the extension using PGXS and PGXN, and how to *troubleshoot* common issues that might arise during the build and installation process.

By the end of this section, novice developers should be able to understand and use PGXS and PGXN to build, install, and distribute their extensions.

5. **Procedural Languages and External Languages:** These sections will discuss the role of procedural and external languages in extension development and explain how to use them.

Problem:
Novice developers may not be familiar with procedural and external languages or their role in Postgres extension development.

Solution:

- Explain what procedural languages and external languages are, and their role in Postgres extension development.
- Explain how to use procedural languages in extension development, including how to write functions and triggers using
    - PL/pgSQL
    - PL/Python.
- Provide examples of popular external languages used in Postgres extension development, such as
    - JavaScript
    - Python.
- Explain how to use external languages in extension development, including how to write *functions* and *triggers* using external languages, and how to compile and link external code to Postgres.
- Provide code snippets and examples for each language, showing how to write simple functions and triggers using each language.
- Explain how to test and *debug functions* and triggers written in the target language.

6. **Regression Tests:** This section will discuss the importance of regression tests, explain how to write regression tests for an extension, and provide examples of regression tests.

Problem:
Novice developers may not understand the importance of regression testing and may not know how to write regression tests for their extensions.

Solution:
- Explain the importance of regression testing: Emphasize the importance of regression testing to catch bugs before they make it into production and cause problems for end-users.
- Provide a step-by-step guide for writing regression tests for extensions. This guide should include the following steps:
    - ***Set up a test database:*** Create a new database specifically for testing the extension. This database should be separate from the production database.
    - ***Write test cases***: Write test cases that cover all the features of the extension. Each test case should be designed to test a specific feature or scenario.
    - ***Write test scripts:*** Write test scripts in the target language (*Python or JavaScript*) to automate the testing process.
    - ***Run the tests:*** Run the test scripts against the test database to ensure that the extension is working as expected.

- Provide examples of regression tests: To help developers understand how to write regression tests, provide code examples of regression tests in the target language.

7. **Release Management and Upgradability:** This section will discuss best practices for managing releases and upgrades of extensions, as well as how to make an extension upgradable.

To ensure that the tutorial and sample code are accurate and up-to-date, we will **collaborate** with other Postgres developers to **incorporate feedback** and suggestions for improving the extension development process. We will also provide ongoing support and maintenance for the tutorial and sample code, including updating it to reflect changes in Postgres and addressing feedback from users.

Approach for completion of the Postgres Extension Tutorial:

As a beginner in Postgres, I bring a *fresh perspective* to the Postgres Extension Tutorial and Quick Start Guide project. My position affords me the opportunity to *experience first-hand* the challenges that novice programmers encounter when developing or utilizing extensions.

My status as a relative newcomer to the topic of Postgres Extensions will allow me to approach the problem with a *beginner's mindset*, unencumbered by preconceived notions or assumptions about the process. This unique position enables me to provide valuable *insight* into the *difficulties* that others may face when attempting to navigate the *Postgres extension system*.

1. Planning Phase:

- The first phase of the project will involve planning and scoping the work required.
    - This will involve a *detailed review* of the existing documentation for the Postgres extension system and identifying areas that need improvement.
    - We will also identify the programming languages and tools required for the project, as well as any dependencies or third-party libraries needed.

2. Development Phase:

- The development phase will involve creating the tutorial and sample code, as well as the testing framework for Postgres extensions.
    - The tutorial will be written in a *clear* and *concise* language and structured in a way that is easy for developers to follow.
    - Sample code will be provided for each topic covered in the tutorial, demonstrating best practices for extension development.
    - The *testing framework* will be designed to allow developers to easily create and run regression tests for their extensions.

3. Collaboration and Feedback:

- Throughout the development phase, we will *collaborate* with other Postgres developers to *incorporate feedback* and suggestions for improving the extension development process.
- This will involve sharing the tutorial and sample code with the Postgres community and *actively seeking feedback* through online forums and mailing lists.

4. **Documentation Improvements:**
   - As part of the project, we will also improve the existing documentation for the Postgres extension system.
     - This will involve updating the official Postgres documentation with new content and clarifying existing content to make it more accessible to developers. Following the *Postgres docguide* which specifies how the documentation is formatted (https://www.postgresql.org/docs/current/docguide.html).

5. **Best Practices and Guidelines:**
   - We will create a set of best practices and guidelines for extension development, based on our experience and feedback from the Postgres community.
   - These best practices will cover areas such as:
     - code structure
     - naming conventions
     - release management.

6. **Support and Maintenance:**
   - Once the tutorial and supporting resources are complete, we will provide ongoing support and maintenance for the project.
     - This will involve updating the tutorial and sample code to reflect changes in Postgres and addressing feedback from users.
     - We will also provide support to developers who are using the tutorial and testing framework, answering questions and providing assistance as needed.

Approximate schedule:

Timeline	Deliverables
Week 1 (5/29-6/4)	**Planning and scoping:** • Review existing Postgres extension system documentation
Week 2 (6/5-6/11)	**Planning and scoping:** • Identify areas for improvement and develop a plan for the tutorial • Define project milestones and deliverables
Week 3 (6/12-6/18)	**Development phase – 1:** • Set up development environment
Week 4 (6/19-6/25)	**Development phase – 1:** • Develop the tutorial framework and structure • Write and test sample code for each topic covered in the tutorial
Week 5 (6/26-7/2)	**Collaboration and Feedback:** • Share the tutorial and sample code with the Postgres community for feedback
Week 6 (7/3-7/9)	**Collaboration and Feedback:** • Incorporate feedback and suggestions for improvement • Update the tutorial and sample code accordingly
Week 7 (7/10-7/16)	**Development phase – 2:** • Develop the testing framework for Postgres extensions
Week 8 (7/17-7/23)	**Development phase – 2:** • Write and test regression tests for sample code
Week 9 (7/31-8/6)	**Documentation Improvements:** • Improve the official Postgres extension system documentation with new content
Week 10 (8/7-8/13)	**Documentation Improvements:** • Clarify existing content to make it more accessible to developers
Week 11 (8/14-8/20)	**Best Practices and Guidelines:** • Develop a set of best practices and guidelines for extension development
Week 12 (8/21-8/26)	**Best Practices and Guidelines:** • Cover areas such as code structure, naming conventions, and release management