



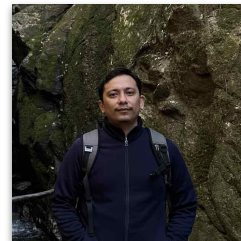
# Extensions in PostgreSQL, and how to develop one

- By, Swastik Gurung



# Speaker's Bio

- **Name:** Swastik Gurung
  - **Designation:** Data Operations Manager @ Bajra Technologies Pvt. Ltd.
  - **Skills:** SQL, PostgreSQL, MySQL, Linux, Python
  - **Hobbies:** Traveling, Hiking, Nature photography, Cooking
  - **Linkedin:** [linkedin.com/in/swastik-gurung-449ab2136/](https://www.linkedin.com/in/swastik-gurung-449ab2136/)
- GitHub:** [github.com/swastik1990](https://github.com/swastik1990)





# Introduction

- Non-core code that can be installed into a database
- Add new functions or features
- Modular approach to new features
- Popular Examples (e.g., `pg_stat_statements`, `PostGIS`, `pgcrypto`)



# Useful Things to Know

- Some extensions are "trusted" and can be created by non-superusers
- Install Extensions objects into a specified schema
- Extension can be updated to a newer version
- Some extensions depend on others



# When to Use Extensions

- Add specialized functionality (like geospatial features or time-series optimization)
- Enhance performance with specialized index types
- Add new data types or operators
- Extend PostgreSQL procedural languages



# Why use extensions

- Better way to bundle together multiple related functions or features
- Easier to install (CREATE EXTENSION), configure and manage extensions
- Modular architecture
- Community or Third-party involvement
- Reusable Code



# Understanding PostgreSQL Extensions

- Built-in (shipped with PostgreSQL) which are Trusted
- Third-Party (developed externally) Extensions, usually Untrusted
- Where to find extensions: contrib modules, PGXN, GitHub, OS package managers
- Extensions are installed per-database, not server-level
- Use cases: performance monitoring (**pg\_stat\_statements**), GIS support (**PostGIS**), encryption (**pgcrypto**)



# Installing and Managing Extensions

- Extensions can be installed via **CREATE EXTENSION <name>;**
- List installed extensions: `\dx` or **SELECT \* FROM pg\_extension;** in `psql`
- To display available extensions: **SELECT \* FROM pg\_available\_extensions ORDER BY name;**
- To Update an extension: **ALTER EXTENSION ... UPDATE;**
- To Remove an extension: **DROP EXTENSION**





# Extension Directory Structure

- SQL Files (`extension--version.sql`)
- Control File (`extension.control`)
- Makefile (compiling with C)
- Shared Libraries and C Code (if applicable)

```
my_extension/  
├── my_extension.control  
├── my_extension--1.0.sql  
├── Makefile  
└── src/  
    └── my_extension.c
```



# Developing a Basic Extension

- Either copy to local extension directory or use make install
- Writing the Control File
- Creating SQL Scripts for Functions, Types, or Tables
- Registering Objects with PostgreSQL
- Write in C to gain performance (*low-level language*)
- Create a GitHub Repo for Community (*optional*)



# Writing Extensions in C

- Use C for performance boost, access to postgres internals
- PostgreSQL Server Programming Interface (SPI) for DB interactions
- Creating Shared Libraries
- Compiling with `pg_config` and `Makefile`



# Versioning and Upgrades

- Naming Convention for Version Scripts. Ex: **extension--1.0--1.1.sql**
- Use **ALTER EXTENSION UPDATE** to upgrade smoothly
- Consider ALTERs over DROPs to preserve user data
- Think of backward compatibility, migration scripts



# Testing and Debugging Extensions

- Unit Testing with **pgTAP**
- Enable Logging and Error Handling, with `log_min_messages`
- Common Debugging Techniques



# Packaging and Distribution

- Add README, LICENSE, Makefile, and metadata files
- Distribute via GitHub or PGXN (PostgreSQL Extension Network)
- Consider making it installable via apt, yum, or Homebrew



# Security Considerations

- Dealing with Privileges (Schema, Table, Function)
- Try to avoid Superuser Privileges
- Use Trusted Sources for Extensions (contrib, PGXN, reputed GitHub Repos)
- Separate/Dedicated schema for extensions
- Avoid dynamic SQL injections and unsafe file operations



# PostgreSQL Extensions (PostGIS)

- Spatial database extender for PostgreSQL
- Allows location queries to be run in SQL
- Supports geometric and geographic types: POINT, LINESTRING, POLYGON, MULTIPOLYGON, etc.
- Hundreds of spatial functions: ST\_Contains, ST\_Intersects, ST\_Distance, ST\_Within, etc.
- Completely free and open source under the GNU GPL license
- **Use Cases:** Storing and querying spatial data (e.g., maps, regions, roads), Finding nearby places, route calculations





## PostgreSQL Extensions (postgres\_fdw)

- Allows querying remote PostgreSQL servers from a local
- Remote tables usable in SQL queries
- Supports SELECT, INSERT, UPDATE, and DELETE on foreign tables
- Uses PostgreSQL backend connections
- **Use Cases:** Federated queries, without ETL



# PostgreSQL Extensions (pg\_stat\_statements)

- Tool for query performance monitoring
- Detailed Metrics Collected: Total calls, Total time spent, Min/Max/Mean execution time, Rows returned, Shared/local block I/O, Temp file usage
- Useful for DBAs to optimize queries, indexes, and app performance
- Works well with tools like pgBadger, pg\_stat\_monitor, Grafana, etc.
- **Use Cases:** Identify slow queries, most frequently executed queries



# PostgreSQL Extensions (pg\_trgm)

- Extension for fuzzy string matching and text search
- Breaks strings into trigrams (three-character chunks) for similarity comparisons
- Enables fuzzy search using %, LIKE, and ILIKE with index support
- Similarity Functions: `similarity(text, text)`, `word_similarity(text, text)`, `show_trgm(text)`
- Can be combined with PostgreSQL's full-text search for advanced use cases
- **Use Cases:** Fuzzy search, Deduplication



# PostgreSQL Extensions (pgvector)

- Vector storage in native PostgreSQL format
- Indexing support for fast similarity search
- Supports cosine, inner product, and L2 distance
- Seamless integration with AI models and pipelines
- **Use Cases:** Semantic search, Recommendation systems, AI/ML applications



# PostgreSQL Extensions (pgcrypto)

- Cryptographic functions for data encryption, decryption, hashing, and digital signatures
- Works inside SQL, so we can encrypt/decrypt data in queries
- Storing encrypted sensitive data (like SSNs, emails, etc.)
- **Use Cases:** Data Encryption, Password hashing



# Conclusion

- Extensions unlock PostgreSQL's true power
- Start small—build SQL-only extensions, move to C if needed
- Contribute to the community: share your work!



## Hands On Exercises



## References

PostgreSQL Documentation | CREATE EXTENSION [Link](#)

10 PostgreSQL extensions you need to know about | PostgreSQL 101 [Link](#)

Software Catalogue - PostgreSQL extensions [Link](#)

Top 8 PostgreSQL Extensions [Link](#)