

# PostgreSQL Extensibility

*Jeff Davis,  
Truviso, Inc.*

# Introduction

- Many businesses have needs that are specific to the industry or to the business itself
  - Not always met by generic tools offered by a DBMS
- People in that business know best how to solve those problems
- But those people may not be database engine developers

# And Also...

When implementing a workaround on a running system, you want every possible option available.

# Outline

- How to solve domain-specific problems
  - Solutions should be as seamless as a native feature, not bolted on.
- How to make those solutions perform like a special-purpose solution
- How to build complete features without modifying the core database engine
- Where these features fit into the PostgreSQL development landscape

# Extensibility Overview

- Functions – procedural code that can be written in a variety of languages
- Operators – define the behavior of your type and allow it to be usefully indexed
- Types – input/output
- Index Support – define sophisticated indexing behavior for your data type
- Procedural languages

# Functions

- Can be written in a variety of languages
  - PL/pgSQL, C, Perl, Python, and more
- Fundamental part of extensibility architecture
- Used for
  - Operators
  - Type input/output functions
  - Index support routines
  - etc.

# Operators

- Allowable characters
  - + - \* / < > = ~ ! @ # % ^ & | ` ?
- Can be infix, postfix, or prefix
- Allow index searches

# Types

- Types help you represent data the way that makes sense to your business
- User-defined types are given all the power of native types
  - Not slower
  - Not less flexible
  - Not less indexable
- Define input/output, storage, etc.



# Index Access Methods

- BTree
- GiST
- GIN
- You implement your specialized indexing algorithm using index support functions
- You don't need to worry about low level details like page splits or the transaction log (WAL)

# Example: PostGIS

- Spatial data – implements OpenGIS
  - Collections of lines, points, polygons, etc.
- Important queries
  - Contains
  - Overlaps
- Existing data types can't represent the data effectively
- BTree can't index effectively, no useful total order for spatial types

# PostGIS: Types

- Geometry type can hold
  - points
  - linestrings
  - polygons
  - etc.
- A single geometry value can hold a combination of all of those

# PostGIS: operators

- && - overlaps
- << - strictly left of
- >> - strictly right of
- ~= - is the same as
- ~ - contains
- @ - is contained by

# PostGIS: indexes

- Uses GiST, a sophisticated generalized index access method with a tree structure.
- GiST is much more general than BTree. It can search for matches based on what makes sense for your problem, using algorithms that you supply via functions.
- Still has all the safety of the write-ahead log.

# Example: DBI-Link

- Allows you to access any data source available via Perl's DBI
- Remote data is seen as a table in PostgreSQL
- How?

# DBI-Link: PL/Perlu

- PL/Perlu is the “untrusted” (i.e. not sandboxed) version of the perl procedural language.
- Can execute arbitrary code, including creating sockets, etc.
- And therefore, it can use Perl's DBI to access any data over a network.

# DBI-Link: SRFs

- DBI-Link uses Set Returning Functions, or SRFs
- Function returns multiple rows to PostgreSQL
- Is used in place of a table.



# Example: PL/proxy

- Access remote PostgreSQL data
- Designed to be easy to retrieve data from many different PostgreSQL nodes
- Supports partitioning
- Implemented as its own language

# PL/proxy: example

```
CREATE OR REPLACE FUNCTION
    get_user_email(i_username text)
RETURNS SETOF text AS $$
    CLUSTER 'usercluster';
    RUN ON hashtext(i_username) ;
    SELECT email FROM users
        WHERE username = i_username;
$$ LANGUAGE plproxy;
```

# Example: Full Text Search

- Search for matches within a document.
- The primary query is “contains”
- Problem: BTree unsuitable, documents do not have a useful total order.
- Problem: Text type is unsuitable
  - Lists the same word many times, doesn't eliminate stopwords
  - Doesn't equate words with same root

# Full Text Search: Types

- `tsquery` – type suitable for representing queries
  - `to_tsquery('bears & eat')`
  - `'bear' & 'eat'`
- `tsvector` – document representation suitable for indexed searches
  - `to_tsvector('Bears eat everything.')`
  - `'eat':2 'bear':1 'everyth':3`

# Full Text Search: Indexes

- Indexable Operator - @@
  - Means “matches the query”
- Index Access methods
  - GIN – better query performance and scalability
  - GiST – better update performance
- GiST is such a generic index interface, it can be used for indexing spatial data and full text data

# Full Text Search: 8.3

- Full Text Search lived as a fully functional separate module, including index support, since 2002.
- Was very successful and useful to many people
- Included in PostgreSQL 8.3
- The fact that such a major feature was fully functional completely outside of the core PostgreSQL product shows the power of the extensible architecture.

# Development Landscape

- Variety of projects outside of PostgreSQL core
  - <http://www.pgfoundry.org>
- Advantages
  - Separate release cycle
  - Separate development
  - Users can pick and choose what fits them best among alternative solutions
- Disadvantages
  - Can't extend SQL language

# Conclusion

- Extensibility allows people with knowledge in specific problem domains to create major features without dealing with database internals.
- More efficient development by making independent projects
- Enables businesses to solve practical problems quickly and cheaply, without extensive error-prone “glue” code.
- Doesn't require interrupting normal operation!