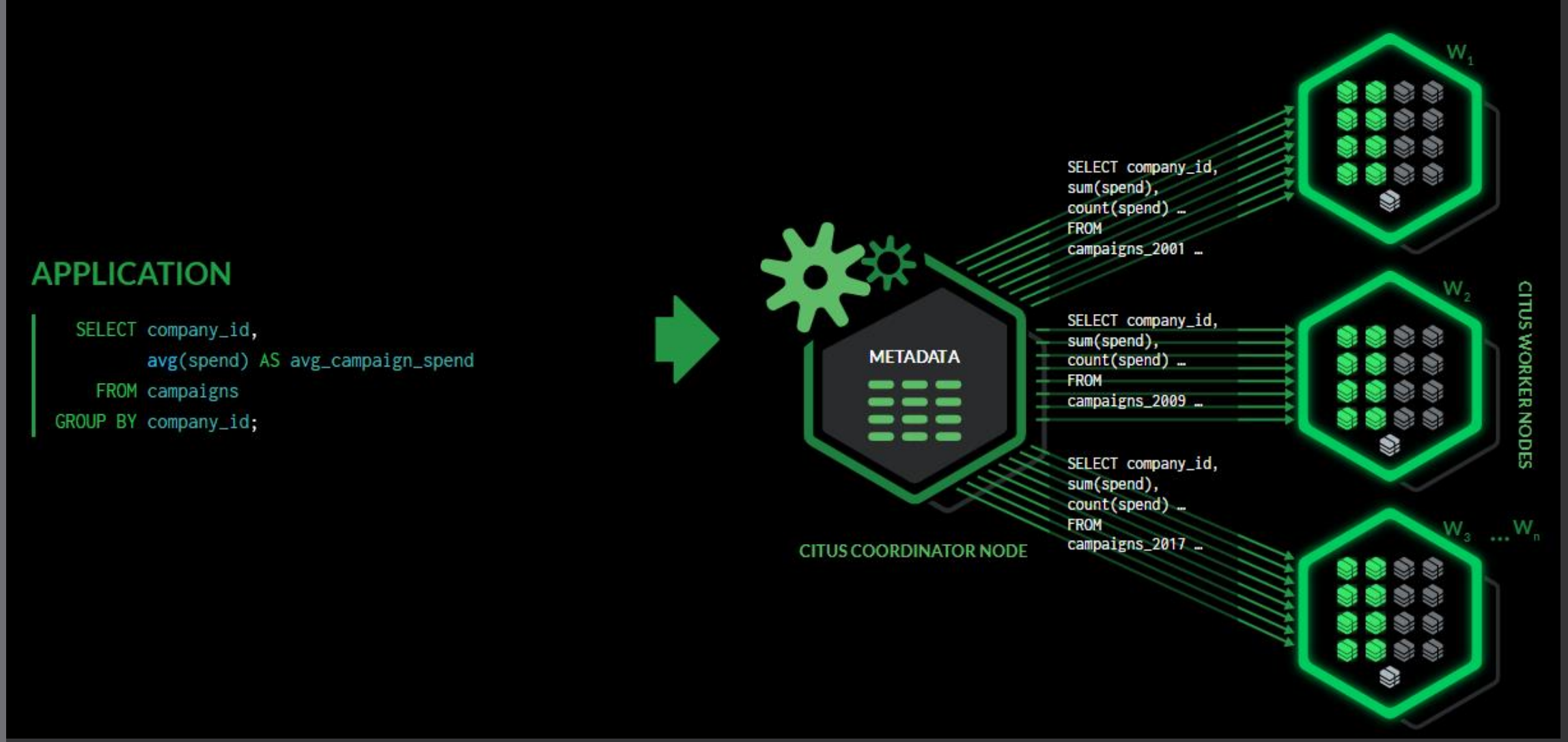
Implementing your first PostgreSQL extension: From Coding to Distribution

Burak Yücesoy

Önder Kalacı

Who are we?

- Two engineers from Microsoft.
- Used to work for Citus Data (now acquired by Microsoft).
- Earn our lives by developing PostgreSQL extensions.


APPLICATION
SELECT company_id,
avg(spend) AS avg_campaign_spend
FROM campaigns
GROUP BY company_id;
METADATA
CITUS COORDINATOR NODE
SELECT company_id,
sum(spend),
count(spend) …
FROM
campaigns_2001 …
SELECT company_id,
sum(spend),
count(spend) …
FROM
campaigns_2009 …
SELECT company_id,
sum(spend),
count(spend) …
FROM
campaigns_2017 …
W1
W2
W3 … Wn
CITUS WORKER NODES

Who are we?

Outline

- What is PostgreSQL Extension?
- Why PostgreSQL is extendable?
- What is extendable in PostgreSQL?
- Life cycle of PostgreSQL extension development
  - Developing
  - Debugging
  - Testing
  - Packaging
  - Releasing

What is PostgreSQL Extension?

- A PostgreSQL extension is a piece of software that adds functionality to Postgres. Each extension bundles related objects together.
- Postgres 9.1 started providing official APIs to override or extend any database module's behavior.
- "`CREATE EXTENSION extension_name;`" dynamically loads these objects into Postgres' address space.

Example Extension: pg_cron

- A task scheduler inside the database; it allows to perform periodic jobs on the database.
- Every Saturday at 03:30; delete old data
  - psql> SELECT cron.schedule('30 3 * * 6', $$DELETE FROM events WHERE event_time < now() - interval '1 week'$$);
- Every day at 10:00, run VACUUM
  - psql> SELECT cron.schedule('0 10 * * *', $$VACUUM$$);

Why PostgreSQL is Extendable?

- Every decade brings new workloads for databases.
- The last decade was about capturing more data, in more shapes and form.
- Postgres has been forked by dozens of commercial databases for new workloads. When you fork, your database diverges from the community.
- What if you could leverage the database ecosystem and grow with it?

What is Extendable in PostgreSQL?

You can override, cooperate with, or extend any combination of the following database modules:

- Type system and operators
- User defined functions and aggregates
- Storage system and indexes
- Write ahead logging and replication
- Transaction engine
- Background worker processes
- Query planner and query executor
- Configuration and database metadata

Type system and operators

- PostgreSQL already has lots of different data types;
  - bigint, text, timestampz, jsonb, …
- If you need a data type which doesn't exist in PostgreSQL;
  - You can define new type with **CREATE TYPE** command.
  - You can add the types created by other people using extensions
- Some additional data types;
  - ip-address, e-mail

Type system and operators

```
CREATE TYPE name AS
    ( [ attribute_name data_type [ COLLATE collation ] [, ... ] ] )

CREATE TYPE name AS ENUM
    ( [ 'label' [, ... ] ] )

CREATE TYPE name AS RANGE (
    SUBTYPE = subtype
    [ , SUBTYPE_OPCLASS = subtype_operator_class ]
    [ , COLLATION = collation ]
    [ , CANONICAL = canonical_function ]
    [ , SUBTYPE_DIFF = subtype_diff_function ]
)

CREATE TYPE name (
    INPUT = input_function,
    OUTPUT = output_function
    [ , RECEIVE = receive_function ]
    [ , SEND = send_function ]
    [ , TYPMOD_IN = type_modifier_input_function ]
    [ , TYPMOD_OUT = type_modifier_output_function ]
    [ , ANALYZE = analyze_function ]
    [ , INTERNALLENGTH = { internallength | VARIABLE } ]
    [ , PASSEDBYVALUE ]
    [ , ALIGNMENT = alignment ]
    [ , STORAGE = storage ]
    [ , LIKE = like_type ]
    [ , CATEGORY = category ]
    [ , PREFERRED = preferred ]
    [ , DEFAULT = default ]
    [ , ELEMENT = element ]
    [ , DELIMITER = delimiter ]
    [ , COLLATABLE = collatable ]
)

CREATE TYPE name
```

Type system and operators

- It is also possible to define (or overwrite) operators for the types you created.
- For example > or < operators can be meaningful for ip address data type.
- Or you can come up with completely new operator such as;
  - Distance function for points; `Point <-> Point`
  - Membership of point in Sphere; `Point <.> Sphere`

User defined functions and aggregates

- You can create new function or aggregate using CREATE FUNCTION command.
- If you are performing some operations frequently it may make sense to implement them as function.
- Also if you defined new type, you can also create the functions to perform specific things on the type you create.

User defined functions and aggregates

```
CREATE [ OR REPLACE ] FUNCTION
    name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_expr ] [, ...] ] )
    [ RETURNS rettype
      | RETURNS TABLE ( column_name column_type [, ...] ) ]
  { LANGUAGE lang_name
    | TRANSFORM { FOR TYPE type_name } [, ... ]
    | WINDOW
    | IMMUTABLE | STABLE | VOLATILE | [ NOT ] LEAKPROOF
    | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
    | PARALLEL { UNSAFE | RESTRICTED | SAFE }
    | COST execution_cost
    | ROWS result_rows
    | SET configuration_parameter { TO value | = value | FROM CURRENT }
    | AS 'definition'
    | AS 'obj_file', 'link_symbol'
  } ...
```

What is Extendable in PostgreSQL?

You can override, cooperate with, or extend any combination of the following database modules:

- Type system and operators
- User defined functions and aggregates
- ~~Storage system and indexes~~
- Write ahead logging and replication
- Transaction engine
- Background worker processes
- Query planner and query executor
- Configuration and database metadata

Can be done in SQL

Needs to be done in lower lever like C

Developing

Debugging

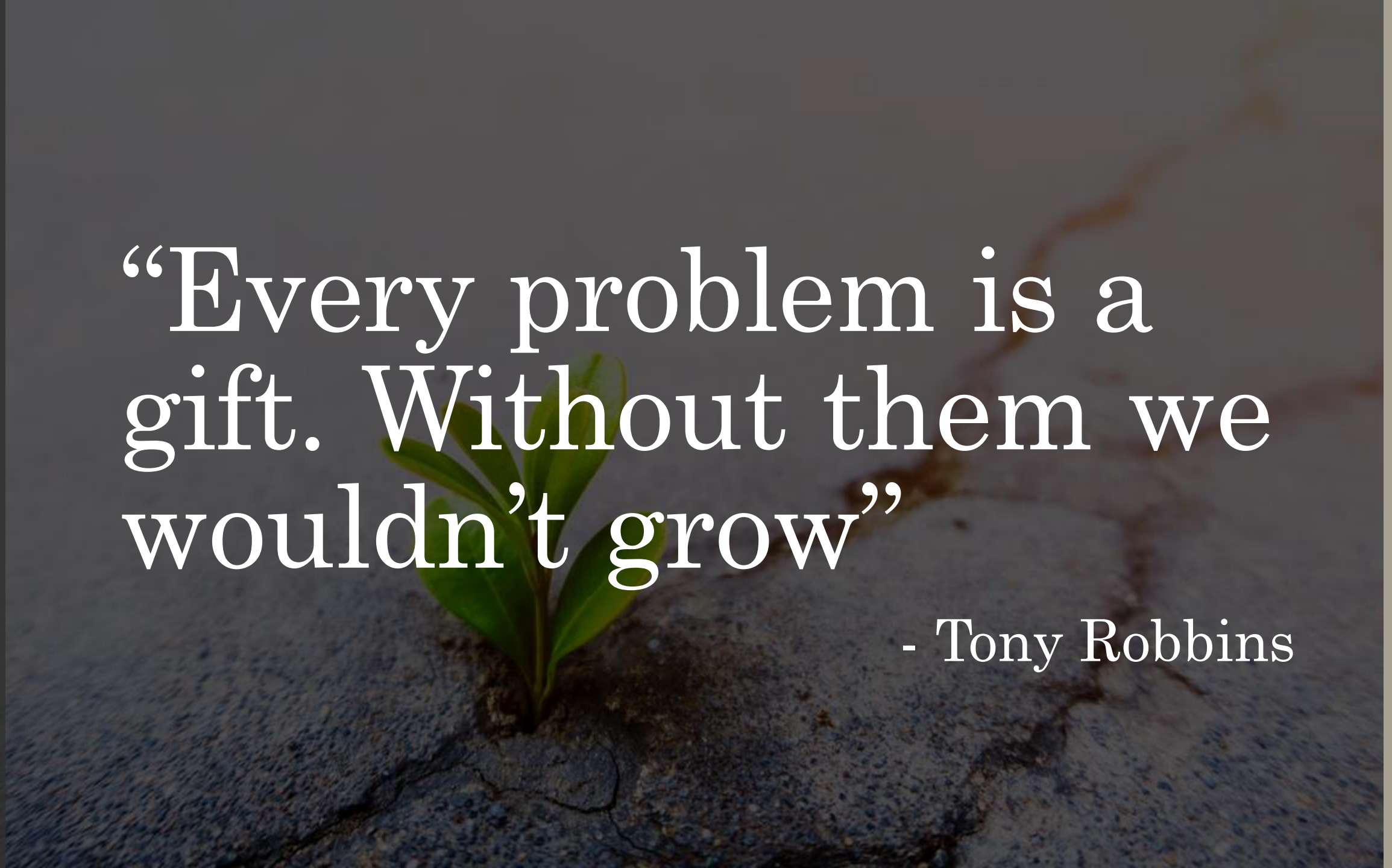
Testing

Packaging

Releasing

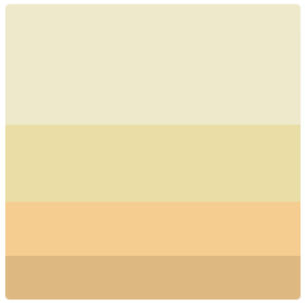
PostgreSQL Extension Development Life Cycle

Disclaimer


“Every problem is a gift. Without them we wouldn’t grow”
- Tony Robbins

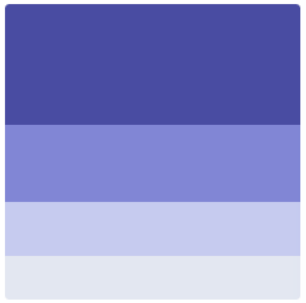
Color Hunt Palettes › New

106 Today



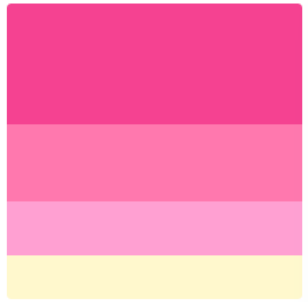
102 Yesterday

130 2 days



151 3 days

99 4 days



249 5 days

353 6 days

425 1 week

341 1 week

352 1 week

Search Palettes

Color Palettes for Designers and Artists

Color Hunt is a free and open platform for color inspiration with thousands of trendy hand-picked color palettes

Get our Chrome extension for color inspiration in every new tab

Add to Chrome

Made with ♥ by Gal Shir

Today's Schedule for Extension Development

- Today we will follow several steps to create a working prototype to represent color information in database;
  - Primitive approach with using text
  - Composite type
  - UDFs and operators for our type
  - C level implementation
  - Creation of custom nodes for our type
  - Modifications to executor and planner

- You can follow the development from this repo

PostgreSQL Internals: Datums

- Datums are PostgreSQL's way of representing single value.
- Values are passed to or from PostgreSQL as Datums.
- It encapsulates the actual value and provides generic interface for all kinds of value types.
- The code using the Datum has to know which type it is, since the Datum itself doesn't contain that information.
- Conversion to and from Datum is made by helper functions;
  - Int32GetDatum(int): Converts int value to Datum
  - DatumGet32Int(Datum): Gets the int value stored in Datum

PostgreSQL Internals: Tuple

- Tuples have many different use-cases but most importantly are representation of rows in the database.
- They are made up from Datums.
- Interactions with tuples are made by macros defined in PostgreSQL codebase

PostgreSQL Internals: Memory Context

- All memory allocations is handled by various memory contexts.
- You need to allocate memory by palloc() function instead of standard malloc().
- There are multiple memory contexts with different lifetimes;
  - TopMemoryContext
  - CacheMemoryContext
  - MessageMemoryContext
- More information at: https://github.com/postgres/postgres/blob/master/src/backend/utils/mmgr/README

PostgreSQL Internals: Error Reporting

- elog and ereport functions are used for error reporting.
- They are used to print user visible error messages, but more importantly;
  - They rollback open transaction
  - They release any allocated memory for the transaction/queries in related memory contexts.
- It is even possible to extend the way error messages are handled in PostgreSQL
- More information at: https://github.com/postgres/postgres/blob/master/src/backend/utils/error/elog.c#L3

PostgreSQL Internals: Node

- PostgreSQL creates a query text to a query tree.
- Query tree is made up from nodes.
- Each node has a type and related data in it.
- It is possible to create your own node types.

- More information: https://github.com/postgres/postgres/blob/master/src/backend/nodes/README

Let's Implement An Equality Function

```
Datum

color_eq(PG_FUNCTION_ARGS)

{









}
```

Let's Implement An Equality Function

```
Datum

color_eq(PG_FUNCTION_ARGS)

{

  color *c1 = PG_GETARG_COLOR(0);









}
```

Let's Implement An Equality Function

```
Datum

color_eq(PG_FUNCTION_ARGS)

{

  color *c1 = PG_GETARG_COLOR(0);

  color *c2 = PG_GETARG_COLOR(1);




}
```

Let's Implement An Equality Function

```
Datum
color_eq(PG_FUNCTION_ARGS)
{
  color *c1 = PG_GETARG_COLOR(0);
  color *c2 = PG_GETARG_COLOR(1);

  return c1->r == c2->r && c1->g == c2->g && c1->b == c2->b;
}
```

Let's Implement An Equality Function V2

```
static bool
EqualPgColorExtendedNode(
    const struct ExtensibleNode *target_node,
    const struct ExtensibleNode *source_node)
{









}
```

Let's Implement An Equality Function V2

```
static bool
EqualPgColorExtendedNode(
    const struct ExtensibleNode *target_node,
    const struct ExtensibleNode *source_node)
{
        PgColorExtendedNode *targetPlan = (PgColorExtendedNode *) target_node;









}
```

Let's Implement An Equality Function V2

```
static bool
EqualPgColorExtendedNode(
    const struct ExtensibleNode *target_node,
    const struct ExtensibleNode *source_node)
{
        PgColorExtendedNode *targetPlan = (PgColorExtendedNode *) target_node;
        PgColorExtendedNode *sourcePlan = (PgColorExtendedNode *) source_node;




}
```

Let's Implement An Equality Function V2

```
static bool
EqualPgColorExtendedNode(
    const struct ExtensibleNode *target_node,
    const struct ExtensibleNode *source_node)
{
        PgColorExtendedNode *targetPlan = (PgColorExtendedNode *) target_node;
        PgColorExtendedNode *sourcePlan = (PgColorExtendedNode *) source_node;

        return targetPlan->interceptedColor->r == sourcePlan->interceptedColor->r &&
               targetPlan->interceptedColor->g == sourcePlan->interceptedColor->g &&
               targetPlan->interceptedColor->b == sourcePlan->interceptedColor->b;
}
```

Testing

- Extensions can use PostgreSQL's own testing suite
- `Make check` and `make installcheck`
- Runs queries against database and compare the output

Packaging

- PostgreSQL is commonly used in RedHat and Debian bases operation systems.
- For each operating system you want to run your extension on, you need generate binaries in that particular system
- Docker is life saver
- Our open source packaging tools; https://github.com/citusdata/packaging

Packaging

Minimum requirements;

- debian/pgversions
- debian/control.in
- debian/changelog
- debian/copyright
- debian/rules
- debian/compat

Packaging

- > pg_buildext updatecontrol
- > debuild -uc -us -B --lintian-opts --profile debian --allow-root

Releasing

- PostgreSQL community software repositories
- PGXN
- Your own package repository;
  - You can install your package repository to a server and respond install requests from that server.
  - Managed services; packagecloud.io

Thank you
&
Questions