Postgres for Developers

Look what cool things you can do!

By Peter Eisentraut &
Bruce Momjian

Why Postgres Is Cool

- Object-relational
- Developed by engineers
- Open-source development

Transactions DDL

```
BEGIN WORK;

ALTER TABLE customer ADD COLUMN debt_limit NUMERIC(10,2);

ALTER TABLE customer ADD COLUMN creation_date TIMESTAMP WITH TIME ZONE;

ALTER TABLE customer RENAME TO cust;

COMMIT;
```

Arrays

```
CREATE TABLE employee (name TEXT PRIMARY KEY, certifications  TEXT[]);

INSERT INTO employee VALUES ('Bill', ' {"CCNA", "ACSP", "CISSP"} ');

SELECT name
FROM employee
WHERE certifications @> '{ACSP}';

 name
------
 Bill
```

Range Types

```
CREATE TABLE car_rental (id SERIAL PRIMARY KEY, time_span  TSTZRANGE);

INSERT INTO car_rental
VALUES (DEFAULT, '[2016-05-03 09:00:00, 2016-05-11 12:00:00) ');

SELECT * FROM car_rental
WHERE time_span @> '2016-05-09 00:00:00'::timestamptz;

 id | time_span
----+-----------------------------------------------------
  1 | ["2016-05-03 09:00:00-04","2016-05-11 12:00:00-04")
```

Exclusion Constraints

```
CREATE TABLE car_rental (
    id SERIAL PRIMARY KEY,
    car_id INT,
    time_span TSTZRANGE,
    EXCLUDE USING gist (car_id WITH =, time_span WITH &&)
);

INSERT INTO car_rental VALUES (DEFAULT, 1, '[2016-05-03 09:00:00,  2016-05-11
12:00:00)');

INSERT INTO car_rental VALUES (DEFAULT, 1, '[ 2016-05-10 09:00:00, 2016-05-15
12:00:00)');

ERROR:  conflicting key value violates exclusion constraint
"car_rental_car_id_time_span_excl"
```

JSON

```
CREATE TABLE customer (id SERIAL, data  JSONB);

INSERT INTO customer VALUES (DEFAULT, ' {"name" : "Bill", "age" : 21} ');

SELECT data->>'name'
FROM customer
WHERE data @> '{"age" : 21}'::jsonb;

 ?column?
----------
 Bill
```

Expression Indexes

```
CREATE INDEX i_customer_lower ON customer ( lower(name));

SELECT * FROM customer WHERE  lower(name)  = 'cust999';
```

Partial Indexes

```
CREATE INDEX i_customer_name_az ON customer (name)  WHERE state = 'AZ';

SELECT * FROM customer WHERE name = 'cust975' AND  state = 'AZ';
```

Full Text Search

```
SELECT line
FROM fortune
WHERE to_tsvector('english', line)  @@ to_tsquery('cat & (sleep | nap)' );

                                line
-------------------------------------------------------------------
 People who take cat naps don’t usually sleep in a cat’s cradle.
 Q: What is the sound of one  cat napping
```

Trigram Searches

```
SELECT line
FROM fortune
WHERE line ILIKE '%verit%'
ORDER BY 1;

                                    line
-------------------------------------------------------------------------
 body. There hangs from his belt a  veritable arsenal of deadly weapons:
 In wine there is truth (In vino  veritas).
 Passes wind, water, or out depending upon the se verity of the
```

Data Warehouse

- Aggregates
- Optimizer
- Server-side languages, e.g. PL/R
- Window functions
- Bitmap heap scans
- Tablespaces
- Data partitioning
- Materialized views
- Common table expressions (CTE)
- BRIN indexes
- GROUPING SETS, ROLLUP, CUBE
- Parallelism
- Sharding (in progress)

Object Relational

User-defined:

- Aggregates
- Types
- Operators
- Languages
- Casts
- Functions

Everything works together.

Extensions

```
CREATE EXTENSION isn;

\dT

                        List of  data types

 Schema | Name    | Description
--------+---------+-----------------------------------------------
 public | ean13   | International European Article Number (EAN13)
 public | isbn    | International Standard Book Number (ISBN)
 public | isbn13  | International Standard Book Number 13 (ISBN13)
...
```

PostGIS

```
CREATE EXTENSION postgis;

SELECT st_distance(geom, 'SRID=3005;POINT(1011102 450541)') AS distance,
       parcel_id, address
FROM parcels
ORDER BY geom <-> 'SRID=3005;POINT(1011102 450541)'
LIMIT 10;
```

Server-Side Languages

- PL/pgSQL (like PL/SQL)
- PL/Perl
- PL/Python
- PL/Tcl
- SPI (C)
- PL/Java
- PL/Lua
- PL/PHP
- PL/R (like SPSS)
- PL/Ruby
- PL/Scheme
- PL/sh
- PL/v8

Server-Side Languages: PL/Perl

```
CREATE EXTENSION plperlu;

CREATE OR REPLACE FUNCTION email_name(email text) RETURNS text
LANGUAGE plperlu
AS $$
use Email::Address;

my @addresses = Email::Address->parse($_[0]);
return undef unless scalar(@addresses) > 0;
return $addresses[0]->name;
$$;
```

(Another way)

```
CREATE EXTENSION emailaddr;

CREATE TABLE accounts (
    id int PRIMARY KEY,
    name text,
    email emailaddr
);
```

Server-Side Languages: PL/v8

```
CREATE FUNCTION plv8_test(keys text[], vals text[]) RETURNS text
LANGUAGE plv8
IMMUTABLE STRICT
AS $$
    var o = {};
    for(var i=0; i<keys.length; i++){
        o[keys[i]] = vals[i];
    }
    return JSON.stringify(o);
$$;
```

Foreign Data Wrappers

Read and write data from:

- CouchDB
- Informix
- MongoDB
- MySQL
- Neo4j
- Oracle
- Postgres
- Redis
- JDBC
- ODBC
- LDAP
- CSV file

Foreign Data Wrapper Example

```
CREATE EXTENSION mongo_fdw;

CREATE SERVER mongo_server FOREIGN DATA WRAPPER mongo_fdw
    OPTIONS (address '127.0.0.1', port '27017' );

CREATE FOREIGN TABLE warehouse (
    _id name,
    warehouse_id int,
    warehouse_name text,
    warehouse_created timestamptz
) SERVER mongo_server OPTIONS ( database 'db', collection 'warehouse' );

SELECT * FROM warehouse WHERE warehouse_id = 1;
```

Specialized Index Types

- B-tree is ideal for unique values
- BRIN is ideal for the indexing of sorted values, many columns, or large tables
- GIN is ideal for indexes with many duplicates
- SP-GIST is ideal for indexes whose keys have many duplicate prefixes
- GIST for everything else
- Hash indexing is coming back!
- … or write your own

Logical Change Tracking

```
$ pg_recvlogical -d postgres --slot test_slot --create-slot -P wal2json
$ pg_recvlogical -d postgres --slot test_slot --start -o pretty-print=1 -o
write-in-chunks=0 -f -

{
   "change": [
        {
            "kind": "insert",
            "schema": "public",
            "table": "table_with_pk",
            "columnnames": ["a", "b", "c"],
            "columntypes": ["int4", "varchar", "timestamp"],
            "columnvalues": [2, "Tuning", "2015-08-27 16:46:35.818038"]
        }
```

Summary

- Postgres is for developers
- Postgres is extensible
- Postgres is agile
- Postgres is cool