PostgreSQL 12

What is coming up?

Devrim Gündüz

Principal Systems Engineer

@DevrimGunduz

EDB POSTGRES

SELF INTRODUCTION

ME:

- Using PostgreSQL since 1998.
- London PostgreSQL Prime Minister
- “The Guy With The PostgreSQL Tattoo”
- Responsible for PostgreSQL YUM (https://yum.postgresql.org) and ZYPP (https://zypp.postgresql.org) repositories, where we host 200+ software
- PostgreSQL Major Contributor
- Fedora / EPEL packager for 50+ packages.
- London, UK.

SOCIAL MEDIA

- Please follow:
  - @EDBPostgres
  - @PostgreSQL
  - @PGConf_EU

SPECIAL THANKS

- Robert Haas, for many of the slides

SPECIAL THANKS

- Robert Haas, for many of the slides
- Anja, for the great event.

AGENDA

AGENDA

- JIT
- psql improvements
- DBA features
- Table Partitioning
- Indexes
- The Query Planner
- SQL Features
- Odds and Ends
- DoS prevention for some commands
- Postscript: Advanced Server

JIT

JIT

- Now enabled by default

JIT

- Now enabled by default
- auto_explain and EXPLAIN exposes more info about JIT

JIT

- Now enabled by default
- auto_explain and EXPLAIN exposes more info about JIT
- Red Hat / CentOS / Fedora requires postgresql12-llvmjit
- Debian / Ubuntu already bundles in core package.

PSQL IMPROVEMENTS

psql improvements

- Add options for procedures in \df

psql improvements

- Add options for procedures in \df
- Show IP addresses in \conninfo
  - $ psql -p 5412 -h localhost -U postgres -c "\conninfo"
  - You are connected to database "postgres" as user "postgres" on host "localhost" (address "127.0.0.1") at port "5412".
  - $ psql -p 5412 -U postgres -c "\conninfo"
  - You are connected to database "postgres" as user "postgres" via socket in "/var/run/postgresql" at port "5412".

psql: Improvements

- Add options for procedures in \df
- Show IP addresses in \conninfo
  - $ psql -p 5412 -h localhost -U postgres -c "\conninfo"
  - You are connected to database "postgres" as user "postgres" on host "localhost" (address "127.0.0.1") at port "5412".
  - $ psql -p 5412 -U postgres -c "\conninfo"
  - You are connected to database "postgres" as user "postgres" via socket in "/var/run/postgresql" at port "5412".
- Useful if hostname resolves to multiple IP addresses

psql: URLs in HELP

```
[postgres] # \help CREATE DATABASE
Command:     CREATE DATABASE
Description: create a new database
Syntax:
CREATE DATABASE name
    [ [ WITH ] [ OWNER [=] user_name ]
           [ TEMPLATE [=] template ]
      <trrimmed>
URL: https://www.postgresql.org/docs/12/sql-createdatabase.html
```

psql: Tab complete improvements

- ALTER DATABASE … SET TABLESPACE
- Include partitioned tables in what's offered after ANALYZE.
- Include toast_tuple_target in what's offered after ALTER TABLE ... SET| RESET.
- Include HASH in what's offered after PARTITION BY.
- CREATE TABLE <name> with '(', OF or PARTITION OF.
- CREATE TABLE <name> OF with list of composite types.
- CREATE TABLE name (...) with PARTITION OF, WITH, TABLESPACE, ON

psql: Tab complete improvements

- COMMIT (depending on the presence of a temporary table).
- -CREATE TABLE ON COMMIT with actions (only for temporary tables).
- SKIP_LOCKED option for VACUUM and ANALYZE
- ALTER INDEX … ALTER COLUMN … <column number goes here)
- Add completion for storage parameters after CREATE TABLE WITH
- Improve tab completion of ALTER INDEX/TABLE with SET STATISTICS in psql
- ...and a few more.

DBA FEATURES

SKIP_LOCKED for VACUUM and ANALYZE

- Allows VACUUM to skip the work on a relation if there is a conflicting lock on it when trying to open it at the beginning of its processing.

SKIP_LOCKED for VACUUM and ANALYZE

- Allows VACUUM to skip the work on a relation if there is a conflicting lock on it when trying to open it at the beginning of its processing.
- Note: v11: VACUUM can process multiple tables

SKIP_LOCKED for VACUUM and ANALYZE

- Allows VACUUM to skip the work on a relation if there is a conflicting lock on it when trying to open it at the beginning of its processing.
- Note: v11: VACUUM can process multiple tables
- vacuumdb also now has--skip-locked.

SKIP_LOCKED for VACUUM and ANALYZE

- [pagila] # VACUUM (FULL, SKIP_LOCKED,VERBOSE) film,language;

  WARNING: skipping vacuum of "film" --- lock not available

  INFO: vacuuming "public.language"

  INFO: "language": found 0 removable, 6 nonremovable row versions in 1 pages

  DETAIL: 0 dead row versions cannot be removed yet.

  CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.07 s.

  VACUUM

vacuumdb: --min-xid-age and --min-mxid-age

- Improves the selectivity of the relations to vacuum or and analyze

vacuumdb: --min-xid-age and --min-mxid-age

- Improves the selectivity of the relations to vacuum or and analyze
- Transaction ID or multixact ID

vacuumdb: --min-xid-age and --min-mxid-age

- Improves the selectivity of the relations to vacuum or and analyze
- Transaction ID or multixact ID
- Chance to prioritize tables to prevent wraparound of one or the other.

vacuumdb: --min-xid-age and --min-mxid-age

- Improves the selectivity of the relations to vacuum or and analyze
- Transaction ID or multixact ID
- Chance to prioritize tables to prevent wraparound of one or the other.
- When used with --table, opportunity to target subset of tables

max_connections and max_wal_senders

- max_wal_senders is no more a part of max_connections

max_connections and max_wal_senders

- max_wal_senders is no more a part of max_connections
- No more blocking of base backups, if there are enough max_wal_senders are available

max_connections and max_wal_senders

- max_wal_senders is no more a part of max_connections
- No more blocking of base backups, if there are enough max_wal_senders are available
- Now it is like autovacuum and bgworkers.

pg_dumpall and pg_dump

- pg_dumpall: Now can exclude a database from pg_dumpall
  - Can be given once
  - The argument can be a pattern including wildcard characters.

pg_dumpall and pg_dump

- pg_dumpall: Now can exclude a database from pg_dumpall
  - Can be given once
  - The argument can be a pattern including wildcard characters.
- pg_dump: Include ALTER INDEX … ALTER COLUMN … SET STATISTICS info

pg_dumpall and pg_dump

- pg_dumpall: Now can exclude a database from pg_dumpall
  - Can be given once
  - The argument can be a pattern including wildcard characters.
- pg_dump: Include ALTER INDEX … ALTER COLUMN … SET STATISTICS info
- pg_dump: Now allows multiple rows per insert
  - Useful to speed up loading data in a different database engine.
  - pg_dump dbname --inserts --rows-per-insert=250 (Default:100)

postgresql.conf updates

- New parameters:
  - ssl_min_protocol_version = 'TLSv1'
  - ssl_max_protocol_version = ''
  - shared_memory_type (Alternatives: nmap, sysv, windows)
  - log_statement_sample_rate = 1 → (0.1 to 1)

postgresql.conf updates

- New parameters:
  - plan_cache_mode = auto (Alternatives: force_generic_plan, force_custom_plan
    - Executor level, not planner level

postgresql.conf updates

- New parameters:
  - plan_cache_mode = auto (Alternatives: force_generic_plan, force_custom_plan
    - Executor level, not planner level
- Updated parameters:
  - autovacuum_vacuum_cost_delay: 200 ms → 2 ms
  - extra_float_digits : 0 → 1
  - jit = off → on

PARTITIONING

PARTITION PRUNING

```
edb=# \d foo
                Table "public.foo"
 Column |  Type   | Collation | Nullable | Default

--------+---------+-----------+----------+---------
 a      | integer |           |          |
 b      | text    |           |          |
Partition key: RANGE (a)
Indexes:
    "foo_a_idx" UNIQUE, btree (a)
Number of partitions: 1000 (Use \d+ to list them.)

edb=# select * from foo where a = 20190625;
    a     |   b
----------+--------
 20190625 | filler

(1 row)
```

PARTITION PRUNING

```
edb=# EXPLAIN UPDATE foo set b = 'modification' WHERE a =
20190625;
                        QUERY PLAN
-----------------------------------------------------------
 Update on foo  (cost=0.42..8.44 rows=1 width=42)
   Update on foo20
   ->  Index Scan using foo20_a_idx on foo20
(cost=0.42..8.44 rows=1 width=42)
         Index Cond: (a = 20190625)

(4 rows)
```

PARTITION PRUNING: RESULTS

Version	SELECT	EXPLAIN UPDATE
v11	33.114 ms	223.592 ms
v12beta	0.432 ms	0.535 ms

FASTER COPY INTO PARTITIONED TABLES

```
rhaas=# \d bar
                Table "public.bar"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 a      | integer |           | not null |
 b      | text    |           |          |
Partition key: HASH (a)
Indexes:
    "bar_pkey" PRIMARY KEY, btree (a)
Number of partitions: 8 (Use \d+ to list them.)

rhaas=# copy bar from '/Users/rhaas/testcase/testbar.csv'
csv; -- 100k rows
Time: 330.384 ms (on v11)
Time: 276.175 ms (on v12beta, ~16% faster)
```

CONCURRENT ATTACH PARTITION

```
rhaas=# select count(*) from foo;

...
rhaas=# create table foo1000 (a int, b text);
CREATE TABLE
rhaas=# alter table foo attach partition foo1000 for values
from (1000000000) to (1001000000);
ALTER TABLE
```

CONCURRENT ATTACH PARTITION

```
rhaas=# select count(*) from foo;

...
rhaas=# create table foo1000 (a int, b text);
CREATE TABLE
rhaas=# alter table foo attach partition foo1000 for values
from (1000000000) to (1001000000);
ALTER TABLE
```

On v11, the ALTER TABLE will block until the query completes.

CONCURRENT ATTACH PARTITION

```
rhaas=# select count(*) from foo;

...
rhaas=# create table foo1000 (a int, b text);
CREATE TABLE
rhaas=# alter table foo attach partition foo1000 for values
from (1000000000) to (1001000000);
ALTER TABLE
```

On v11, the ALTER TABLE will block until the query completes.

On v12, it will not block.

FOREIGN KEYS TO PARTITIONED TABLES

```
rhaas=# \d bar
                Table "public.bar"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 a      | integer |           | not null |
 b      | text    |           |          |
Partition key: HASH (a)
Indexes:
    "bar_pkey" PRIMARY KEY, btree (a)
Number of partitions: 8 (Use \d+ to list them.)

rhaas=# create table bar_details (id serial primary key, a integer
references bar (a), s text);
ERROR:  cannot reference partitioned table "bar"
```

FOREIGN KEYS TO PARTITIONED TABLES

```
rhaas=# \d bar
                Table "public.bar"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 a      | integer |           | not null |
 b      | text    |           |          |
Partition key: HASH (a)
Indexes:
    "bar_pkey" PRIMARY KEY, btree (a)
Number of partitions: 8 (Use \d+ to list them.)

rhaas=# create table bar_details (id serial primary key, a integer
references bar (a), s text);
ERROR:  cannot reference partitioned table "bar"
```

- On v12, the error is gone!

AVOIDING MERGE APPEND

```
v11:

rhaas=# EXPLAIN SELECT * FROM foo ORDER BY a;
                        QUERY PLAN
-----------------------------------------------------
 Merge Append  (cost=475.04..86230275.02 rows=1000001270
width=11)
   Sort Key: foo0.a
   ->  Index Scan using foo0_a_idx on foo0
(cost=0.42..31389.42 rows=1000000 width=11)
   ...
```

AVOIDING MERGE APPEND

```
v12:

rhaas=# EXPLAIN SELECT * FROM foo ORDER BY a;

                        QUERY PLAN
  ------------------------------------------------------
   Append  (cost=425.15..36394030.55 rows=1000001270
  width=11)

     ->  Index Scan using foo0_a_idx on foo0
  (cost=0.42..31389.42 rows=1000000 width=11)
    ...
```

AVOIDING MERGE APPEND: RESULTS

```
rhaas=# SELECT * FROM foo ORDER BY a OFFSET 1000000000;
 a | b
---+---
(0 rows)

Time: 209820.957 ms (03:29.821) (on v11)
Time: 169733.233 ms (02:49.733) (on v12beta,
      about 19% faster)
```

INDEXING IMPROVEMENTS

BTREE INDEX IMPROVEMENTS: TEST SETUP

```
pgbench -i -s 100

CREATE INDEX on pgbench_accounts (filler);

SELECT oid::regclass, pg_relation_size(oid) FROM pg_class
WHERE relname LIKE 'pgbench%' AND relkind = 'i';

pgbench -T 300 -c 8 -j 8 -N

SELECT oid::regclass, pg_relation_size(oid) from pg_class
WHERE relname like 'pgbench%' and relkind = 'i';
```

BTREE INDEX IMPROVEMENTS: RESULTS

Version	Index on “filler”	Index on “aid”
v11 (initial)	1089 MB	214 MB
v11 (final)	1204 MB	240 MB
v12beta (initial)	1091 MB (+0.1%)	214 MB (+0.0%)
v12beta (final)	1118 MB (-7.2%)	214 MB (-10.9%)

REINDEX CONCURRENTLY

REINDEX takes ShareLock on table and AccessExclusiveLock on index, blocking basically all access to the table – everything except prepared queries that don’t use the index in question.

REINDEX CONCURRENTLY

REINDEX takes ShareLock on table and AccessExclusiveLock on index, blocking basically all access to the table – everything except prepared queries that don’t use the index in question.

REINDEX CONCURRENTLY takes ShareUpdateExclusiveLock on both table and index, permitting concurrent reads and writes.

REINDEX CONCURRENTLY

REINDEX takes ShareLock on table and AccessExclusiveLock on index, blocking basically all access to the table – everything except prepared queries that don’t use the index in question.

REINDEX CONCURRENTLY takes ShareUpdateExclusiveLock on both table and index, permitting concurrent reads and writes.

Similar to DROP INDEX CONCURRENTLY + CREATE INDEX CONCURRENTLY.

REINDEX CONCURRENTLY

REINDEX takes ShareLock on table and AccessExclusiveLock on index, blocking basically all access to the table – everything except prepared queries that don’t use the index in question.

REINDEX CONCURRENTLY takes ShareUpdateExclusiveLock on both table and index, permitting concurrent reads and writes.

Similar to DROP INDEX CONCURRENTLY + CREATE INDEX CONCURRENTLY.

Waits for concurrent transactions to end, twice.

REINDEX CONCURRENTLY

REINDEX takes ShareLock on table and AccessExclusiveLock on index, blocking basically all access to the table – everything except prepared queries that don’t use the index in question.

REINDEX CONCURRENTLY takes ShareUpdateExclusiveLock on both table and index, permitting concurrent reads and writes.

Similar to DROP INDEX CONCURRENTLY + CREATE INDEX CONCURRENTLY.

Waits for concurrent transactions to end, twice.

Watch out for invalid indexes if fails or is interrupted.

CREATE INDEX PROGRESS REPORTING (1/2)

```
-[ RECORD 1 ]------+------------------------------
pid                | 66541
datid              | 16384
datname            | rhaas
relid              | 23914
index_relid        | 0
command            | CREATE INDEX
phase              | building index: scanning table
lockers_total      | 0
lockers_done       | 0
current_locker_pid | 0
blocks_total       | 1639345
blocks_done        | 523774
tuples_total       | 0
tuples_done        | 0
partitions_total   | 0
partitions_done    | 0
```

CREATE INDEX PROGRESS REPORTING (2/2)

```
-[ RECORD 1 ]------+------------------------------
pid                | 66541
datid              | 16384
datname            | rhaas
relid              | 23914
index_relid        | 0
command            | CREATE INDEX
phase              | building index: loading tuples in tree
lockers_total      | 0
lockers_done       | 0
current_locker_pid | 0
blocks_total       | 0
blocks_done        | 0
tuples_total       | 100000000
tuples_done        | 10409041
partitions_total   | 0
partitions_done    | 0
```

GiST and SP-GiST Indexes

- GiST indexes now support INCLUDE columns.
- SP-GiST indexes now support K-nearest-neighbor searches.
- GiST, GIN, and SP-GiST indexes now generate less WAL during index creation.
- VACUUM of GiST indexes is now more efficient and can recycle empty leaf pages.

QUERY PLANNER

PLAN CACHE MODE

- Prepared queries can be handled in two ways.
  - Strategy #1: Replan the query each time it's executed for the particular parameter values in use. ("custom plan")
  - Strategy #2: Create a plan that will work with any parameter values and reuse it. ("generic plan")

PLAN CACHE MODE

- Prepared queries can be handled in two ways.
  - Strategy #1: Replan the query each time it's executed for the particular parameter values in use. ("custom plan")
  - Strategy #2: Create a plan that will work with any parameter values and reuse it. ("generic plan")
- By default, PostgreSQL will try to adaptively pick the best strategy.

PLAN CACHE MODE

- If you know better, you can set plan_cache_mode.
  - Typical use: Force custom plans, because the generic plans are worse than the planner thinks.
  - Possible use: Don’t waste any planning time trying to create worthless custom plans.

SUPPORT FUNCTIONS FOR SQL FUNCTIONS

- **v12:**
- rhaas=# explain select * from generate_series(1, 437218) g;

```
                        QUERY PLAN
-----------------------------------------------------------
```

- Function Scan on generate_series g (cost=0.00..4372.18 rows=437218 width=4)

SQL FEATURES

SUPPORT FUNCTIONS FOR SQL FUNCTIONS

- **v11:**

```
rhaas=# EXPLAIN SELECT * FROM
        generate_series(1, 437218) g;
                        QUERY PLAN
-----------------------------------------------------
 Function Scan on generate_series g
(cost=0.00..10.00 rows=1000 width=4)
```

SUPPORT FUNCTIONS FOR SQL FUNCTIONS

- **v12:**

```
rhaas=# EXPLAIN SELECT * FROM
       generate_series(1, 437218) g;
                        QUERY PLAN
-----------------------------------------------------------

 Function Scan on generate_series g  (cost=0.00..4372.18
rows=437218 width=4)
```

GENERATED COLUMNS

- `rhaas=# CREATE TABLE gce (a int, b int, c int GENERATED ALWAYS AS(a + b) stored);`

GENERATED COLUMNS

- `rhaas=# CREATE TABLE gce (a int, b int, c int GENERATED ALWAYS AS(a + b) stored);`

- Column c can’t be manually updated.

GENERATED COLUMNS

- `rhaas=# CREATE TABLE gce (a int, b int, c int GENERATED ALWAYS AS(a + b) stored);`

- Column c can’t be manually updated.

- It will be recomputed after every INSERT/UPDATE.

GENERATED COLUMNS

- `rhaas=# CREATE TABLE gce (a int, b int, c int GENERATED ALWAYS AS(a + b) stored);`

- Column c can’t be manually updated.
- It will be recomputed after every INSERT/UPDATE.
- Easier (but not necessarily faster) than a TRIGGER.

OTHER FEATURES

R.I.P. TO RECOVERY.CONF

- Finally…

R.I.P. TO RECOVERY.CONF

- Finally…
- PostgreSQL now throws an error if recovery.conf is found.

R.I.P. TO RECOVERY.CONF

- Finally…
- PostgreSQL now throws an error if recovery.conf is found.
- (Almost) everything is moved into postgresql.conf

R.I.P. TO RECOVERY.CONF

- Finally…
- PostgreSQL now throws an error if recovery.conf is found.
- (Almost) everything is moved into postgresql.conf
- No more standby_mode

R.I.P. TO RECOVERY.CONF

- Finally…
- PostgreSQL now throws an error if recovery.conf is found.
- (Almost) everything is moved into postgresql.conf
- No more standby_mode
- recovery_target_timeline=latest by default

R.I.P. TO RECOVERY.CONF

- Settings previously stored in recovery.conf are now in postgresql.conf

R.I.P. TO RECOVERY.CONF

- Settings previously stored in recovery.conf are now in postgresql.conf
- Use recovery.signal or standby.signal to trigger recovery or standby mode

R.I.P. TO RECOVERY.CONF

- Settings previously stored in recovery.conf are now in postgresql.conf
- Use recovery.signal or standby.signal to trigger recovery or standby mode
- Your backup management tool (or scripts) will likely need an update.

R.I.P. TO RECOVERY.CONF

- Settings previously stored in recovery.conf are now in postgresql.conf
- Use recovery.signal or standby.signal to trigger recovery or standby mode
- Your backup management tool (or scripts) will likely need an update.
- A few recovery-related parameters can now be changed without restarting the server: archive_cleanup_command, promote_trigger_file, recovery_end_command, and recovery_min_apply_delay.

R.I.P. TO RECOVERY.CONF

- trigger_file → promote_trigger_file
- pg_basebackup -R appends to postgresql.conf
- A reload is enough for these parameters now:
- archive_cleanup_command
- promote_trigger_file
- recovery_end_command
- recovery_min_apply_delay;

ENABLE OR DISABLE CHECKSUMS OFFLINE

```
[rhaas ~]$ pg_ctl stop
waiting for server to shut down.... done
server stopped
[rhaas ~]$ time pg_checksums -e
Checksum operation completed
Files scanned:  6289
Blocks scanned: 10080538
pg_checksums: syncing data directory
pg_checksums: updating control file
Checksums enabled in cluster

real    2m42.120s
user    0m20.674s
sys 1m30.388s
[rhaas ~]$ du -hs $PGDATA
 87G    /Users/rhaas/pgdata
```

TABLE ACCESS METHODS

- PostgreSQL can now support multiple table storage formats, just as we have for years been able to support multiple index formats (hash, btree, gist, etc.).

TABLE ACCESS METHODS

- PostgreSQL can now support multiple table storage formats, just as we have for years been able to support multiple index formats (hash, btree, gist, etc.).
- Currently, the only in-core table storage method is ‘heap’.

TABLE ACCESS METHODS

- PostgreSQL can now support multiple table storage formats, just as we have for years been able to support multiple index formats (hash, btree, gist, etc.).
- Currently, the only in-core table storage method is ‘heap’.
- Expect more choices in a year or two.

TABLE ACCESS METHODS

- PostgreSQL can now support multiple table storage formats, just as we have for years been able to support multiple index formats (hash, btree, gist, etc.).
- Currently, the only in-core table storage method is ‘heap’.
- Expect more choices in a year or two.
- Support for hidden OID columns removed.

SOME OTHER STUFF

- GSSAPI encryption support
- Progress reporting for CLUSTER and VACUUM FULL
- SERIALIZABLE for parallel query

SOME OTHER STUFF

- pg_upgrade can use filesystem cloning.
- Unified logging framework for client tools, including colorization support.

SOME OTHER STUFF

- Avoid some rewrites in “ALTER TABLE … SET DATA TYPE timestamp”
- When the timezone is UTC, timestamptz and timestamp are binary coercible, avoid the table rewrite, and continue to needlessly rewrite any index on an affected column.

SOME OTHER STUFF

- Log PostgreSQL version number on startup
  - 2019-03-14 11:33:39.842 CET [7829] LOG: starting PostgreSQL 12devel on x86_64-pc-linux-gnu, compiled by gcc (GCC) 8.3.1 20190223 (Red Hat 8.3.1-2), 64-bit
  - 2019-03-14 11:33:39.842 CET [7829] LOG: listening on IPv4 address "127.0.0.1", port 5412
  - 2019-03-14 11:33:39.916 CET [7829] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5412"
  - 2019-03-14 11:33:40.059 CET [7829] LOG: listening on Unix socket "/tmp/.s.PGSQL.5412"

SOME OTHER STUFF

- Allow COPY FROM to filter data using WHERE conditions
  - Extends the COPY FROM command with a WHERE condition, which allows doing various types of filtering while importing the data (random sampling, condition on a data column, etc.).
  - Low overhead

SOME OTHER STUFF

- Allow COPY FROM to filter data using WHERE conditions
  - Extends the COPY FROM command with a WHERE condition, which allows doing various types of filtering while importing the data (random sampling, condition on a data column, etc.).
  - Low overhead
- pg_upgrade: --socketdir option
  - This allows control of the directory in which the postmaster sockets are created for the temporary postmasters started by pg_upgrade
  - Useful for long path names.

SOME OTHER STUFF

- Do not log empty incomplete startup packet
- Change "checkpoint starting" message to use "wal"

SOME OTHER STUFF

- pg_stat_statements_reset can now reset statistics specific to a particular user/db/query.
  - Now, it can discard the statistics gathered so far by pg_stat_statements corresponding to the specified userid, dbid, and queryid.
  - If no parameter is specified or all the specified parameters have default value 0, it will discard all statistics as per the old behavior.

ENTERPRISEDB POSTGRES ADVANCED SERVER

What is new in v12?

WHAT IS NEW IN ADVANCED SERVER 12?

- Interval Partitioning
- Compound Triggers
- MEDIAN, LISTAGG
- CAST(MULTISET)
- System View Improvements

QUESTIONS & DISCUSSION

THANK YOU

info@enterprisedb.com
www.enterprisedb.com

EDB POSTGRES