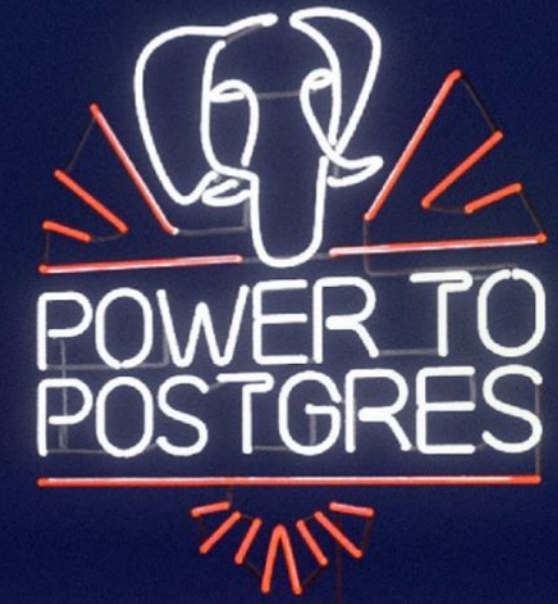
EDB Postgres Advanced Server - What's New?

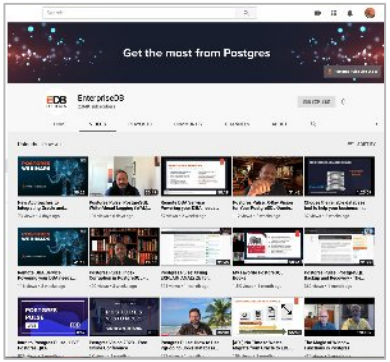
Rushabh Lathia
Marc Linster

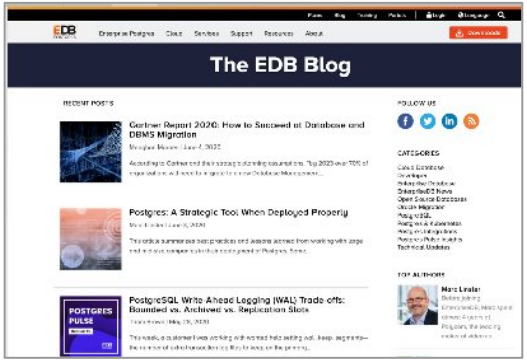


EDB

Agenda

- PG/EPAS v13 key new features
- Deep dive into partitioning
- Q & A session
- Other resources
  - EDB Postgres Youtube Channel
  - EDB Postgres Pulse
  - EDB Postgres Blogs

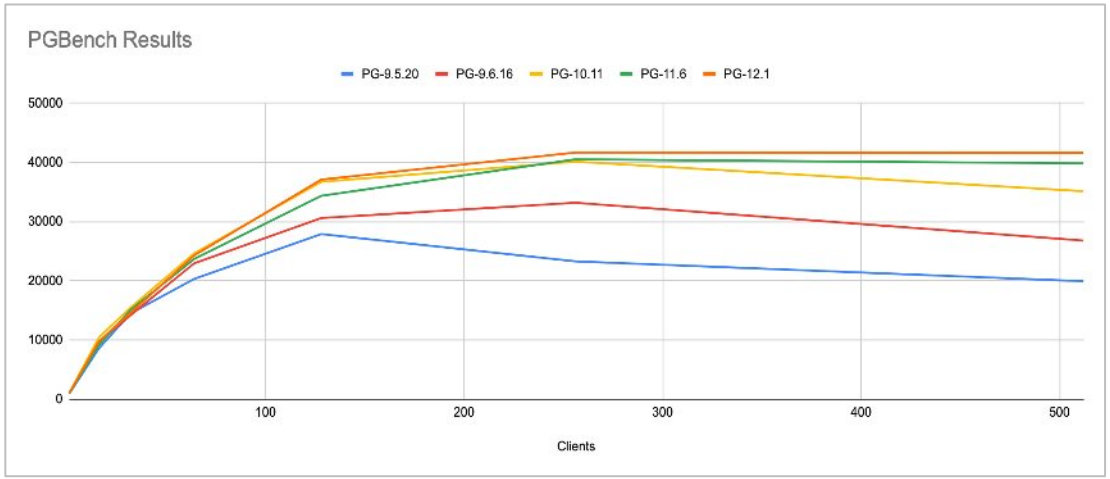




Speed: A key part of Postgre’s evolution

January 2020: 50% TPS improvement in four years

https://www.enterprisedb.com/postgres-tutorials/benchmarking-postgresql-aws-m5metal-instance



Speed is one part of the equation

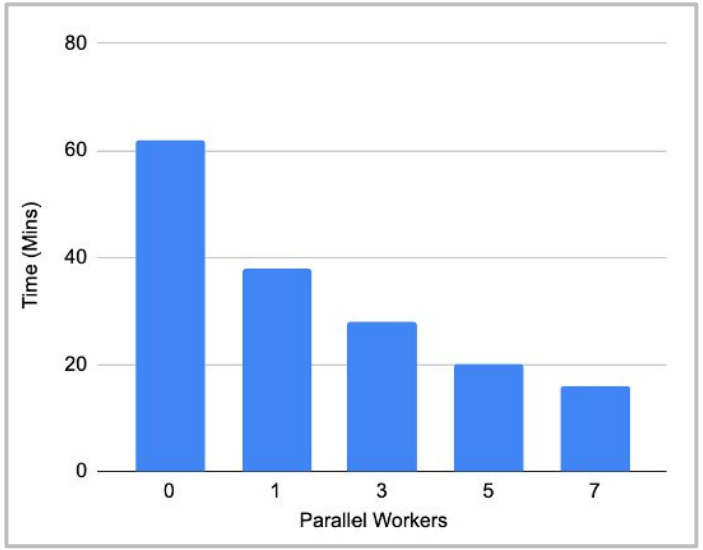
Key new capabilities in EDB Postgres

- Select list of new and enhanced capabilities
  - Vacuum
  - Security and consistency
  - Data Loading
  - Partitioning
- For a complete list, please check out the release notes

Vacuum Improvements

Parallel vacuum of indexes and vacuum for append-only tables

- Performance for parallel vacuum of indexes
  - Vacuum performance after executing 50 million in-place updates - 4X faster in multi process benchmark
- Auto Vacuum for append-only transactions
  - Recalculates statistics!
  - Important for IOT tables



https://www.enterprisedb.com/postgres-tutorials/what-parallel-vacuum-postgresql-13

Security and Consistency

libpq channel binding and improvements of pg_catcheck

- libpq channel binding
  - Stop man in the middle attacks
- New capability for pg_catcheck
  - tool for diagnosing system catalog corruption
  - Find it https://github.com/EnterpriseDB/pg_catcheck
  - New capability: check if the initial file is available for every relation (table)
  - Address ‘could not open file issue’

New capability for pg_catchack

Find “could not open file xxx”

- New option --select-from-relations.
- This option won't tell you the reason why you are getting such errors.
- Works for table, TOAST table, and materialized view in the database.
- This option still doesn't help with indexes, or relation segments other than the first one.
- >>> The relation is accessible <<< “Could not open file”.
- Supports EDB Postgres Advanced Server and PostgreSQL.

Example:

```
rushabh@rushabh:pg_catcheck$
./pg_catcheck edb --select-from-
relations
notice: unable to query relation
"public"."emp": ERROR:  could not
open file "base/16198/16394":
Permission denied
notice: unable to query relation
"public"."jobhist": ERROR:  could
not open file "base/16198/16405":
No such file or directory
progress: done (2
inconsistencies, 0 warnings, 0
errors)
```

EDB*LOADER

edb*loader duplicate row aborts

- Frequently requested feature.
- edbldr used to abort the whole operation if any record insertion failed due to unique constraint violation.
- Fix this by using speculative insertion to insert rows.
- It happens only when 'handle_conflicts' is true and indexes are present.
- Similar to how it is done for "ON CONFLICT ... DO NOTHING". Except, the record goes to the BAD file.

EDB Postgres Partitioning

- PostgreSQL introduced declarative partitioning in v10
  - Getting new features for partitioning with every new release.
  - Performance improvements.
- EPAS had Partitioning feature since 9.4, which was inheritance based.
- From v10, EPAS moved it’s inheritance based partitioning implementation to leverage PostgreSQL declarative partitioning.
  - EPAS adding some cool partitioning features to make a life easy.
  - Redwood compatible syntax for partitioning.
  - Exchange partition.
  - Split Partition.
  - etc..

Partition Automation

Automatically create new partitions when needed w.o. locking

- Easy way to manage the partition scheme.
- Don’t need to worry about partition creation at runtime.
- Easy LOAD or COPY data from old non-partition to partition table.
- Below are the type of automatic partitioning:
    - INTERVAL Partition **(RANGE).**
    - AUTOMATIC Partition **(LIST).**
    - Provide PARTITION and SUB-PARTITION number **(HASH).**

Interval Partition

- INTERVAL partition is an extension to RANGE partition.
- Need to provide an INTERVAL expression for the partition.
- Create a new partition automatically, if given tuple doesn’t fit to the existing partitions.
- INTERVAL clause will decide the range size for a new partition.
- Can also ALTER the existing partition to convert into INTERVAL partition.

Interval Partition

Example (1 of 2)

```
edb@61349=#CREATE TABLE orders (id int, country_code varchar(5), order_date DATE )
PARTITION BY RANGE (order_date) INTERVAL (NUMTOYMINTERVAL(1,'MONTH'))
(PARTITION P1 values LESS THAN (TO_DATE('01-MAY-2020','DD-MON-YYYY')));
CREATE TABLE
edb@61349=#INSERT INTO orders VALUES (1, 'IND', '24-FEB-2020');
INSERT 0 1
edb@61349=#SELECT tableoid::regclass, * FROM orders;
 tableoid  | id | country_code |     order_date
-----------+----+--------------+--------------------
 orders_p1 |  1 | IND          | 24-FEB-20 00:00:00
(1 row)
```

Interval Partition

Example (2 of 2)

```
edb@61349=#INSERT INTO orders VALUES (1, 'IND', '23-JUN-2020');
INSERT 0 1
edb@61349=#SELECT tableoid::regclass, * FROM orders;
      tableoid       | id | country_code |     order_date
---------------------+----+--------------+---------------------
 orders_p1           |  1 | IND          | 24-FEB-20 00:00:00
 orders_sys613490102 |  1 | IND          | 23-JUN-20 00:00:00
(2 rows)
```

Other syntax options:

```
ALTER TABLE orders SET INTERVAL();
ALTER TABLE orders SET INTERVAL(NUMTOYMINTERVAL(1,'MONTH'));
```

Automatic Partition

Automatically create a new partition for a LIST partition

- Automatic list partitioning creates a partition for any new distinct value of the list partitioning key.
- An AUTOMATIC partition is an extension to list partition where system automatically create the partition if the new tuple doesn't fit into existing partitions.
- We can also enable automatic list partitioning on the existing table using the ALTER TABLE command.
- `ALTER TABLE <tablename> SET [MANUAL|AUTOMATIC]`

Automatic Partition Example

Example:

```
CREATE TABLE orders
(id int, country_code varchar(5))
PARTITION BY LIST (country_code)
AUTOMATIC
(PARTITION p1 values ('IND'),
 PARTITION p2 values ('USA'));
```

Describe Table:

```
Partition key: LIST (country_code) AUTOMATIC
Partitions: orders_p1 FOR VALUES IN ('IND'),
            orders_p2 FOR VALUES IN ('USA')
```

Insert a record which doesn't fit in the any of the partition

```
INSERT INTO orders VALUES ( 1 , 'UK');
```

Describe table:

```
Partition key: LIST (country_code) AUTOMATIC
Partitions: orders_p1 FOR VALUES IN ('IND'),
  orders_p2 FOR VALUES IN ('USA'),
  orders_sys15960103 FOR VALUES IN ('UK')
```

Automatic Hash Partitioning

Part 1

- PARTITIONS <num> STORE IN clause"
- allows us to automatically add a specified number of HASH partitions during CREATE TABLE command.
- The STORE IN clause is used to specify the tablespaces across which the partitions should be distributed.

Example:

```
edb@72970=#CREATE TABLE hash_tab (
    col1            NUMBER,
    col2            NUMBER)
 PARTITION BY HASH (col1, col2)
 PARTITIONS 2 STORE IN (tablespace1, tablespace2);
 edb@72970=#¥d hash_tab
       Partitioned table "public.hash_tab"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 col1   | numeric |           |          |
 col2   | numeric |           |          |
Partition key: HASH (col1, col2)
Number of partitions: 2 (Use ¥d+ to list them.)
```

Automatic Hash Partitioning

Part 1

```
edb@89398=#SELECT table_name, partition_name, tablespace_name FROM user_tab_partitions
WHERE table_name = 'HASH_TBL';
 table_name | partition_name | tablespace_name
------------+----------------+-----------------
 HASH_TBL   | SYS0101        | TABLESPACE1
 HASH_TBL   | SYS0102        | TABLESPACE2
(2 rows)
```

Automatic Hash Partitioning

Part 2

- The SUBPARTITIONS <num> creates a predefined template to auto create a specified number of subpartitions for each partition.

```
CREATE TABLE ORDERS  (id int, country_code varchar(5),
order_date DATE)
PARTITION BY LIST (country_code) AUTOMATIC
SUBPARTITION BY HASH(order_date) SUBPARTITIONS 3
( PARTITION p1 VALUES ('USA', 'IND') );

edb@89398=#SELECT table_name, partition_name, subpartition_name
FROM user_tab_subpartitions WHERE table_name = 'ORDERS';
 table_name | partition_name | subpartition_name
------------+----------------+-------------------
 ORDERS     | P1             | SYS0101
 ORDERS     | P1             | SYS0102
 ORDERS     | P1             | SYS0103
(3 rows)
```

Automatic Hash Partitioning

Part 2

```
edb@89398=#INSERT INTO orders VALUES ( 2, 'UK', sysdate );
INSERT 0 1
edb@89398=#SELECT table_name, partition_name, subpartition_name FROM user_tab_subpartitions WHERE table_name = 'ORDERS';
 table_name | partition_name | subpartition_name
------------+----------------+-------------------
 ORDERS     | P1             | SYS0101
 ORDERS     | P1             | SYS0102
 ORDERS     | P1             | SYS0103
 ORDERS     | SYS893980105   | SYS893980106
 ORDERS     | SYS893980105   | SYS893980107
 ORDERS     | SYS893980105   | SYS893980108
(6 rows)
```

Automatic Hash Partitioning

Part 3

- The subpartitions number can be altered using following command and the new number will be used to set up subpartitions in subsequent partitions created

```
ALTER TABLE tbl1 SET SUBPARTITION
TEMPLATE 100;
-- The following will reset the subpartition count to 1
ALTER TABLE tbl1 SET SUBPARTITION
TEMPLATE ( );
```

- The template is used in all partition commands like ADD PARTITION, SPLIT PARTITION where a new partition is created. This template can be overridden by explicit SUBPARTITION description or SUBPARTITIONS <num>

```
ALTER TABLE ORDERS ADD PARTITION p2 VALUES ('AUS') SUBPARTITIONS
10;
¥d orders_p2
 ...
Partition of: orders FOR VALUES IN ('AUS')
Partition key: HASH (col2)
Number of partitions: 10 (Use ¥d+ to list them.)
-- Similar also works when do the SPLIT PARTITION
ALTER TABLE tbl1 SPLIT PARTITION p1 VALUES (10, 20) INTO
(PARTITION p1_a, PARTITION p1_b);
```

Partition Wise Join

Significant Enhancement of existing feature

- PG 11, introduced partition wise join.
- SET enable_partitionwise_join. Default is off.
- Join should be on partition key and both partitions should have same bounds for partitions
- PG 13, enhance the same by removing restriction of having same bounds for partitions.
- For more details about the Declarative Partition Enhancements, the optimizations, and implementation journey, in community, do attend/watch my colleague Amit Langote's talk "Table Partitioning in Postgres, How Far We've Come"

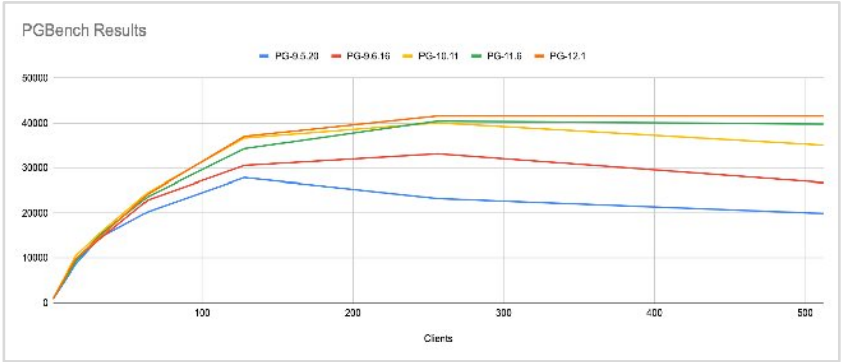
EPAS v13 Features

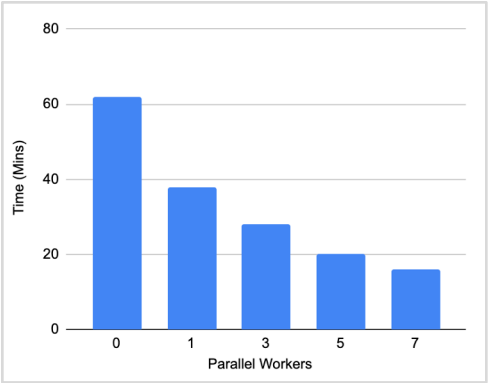
- USING INDEX” clause in the CREATE TABLE and ALTER TABLE.
- STATS_MODE aggregate function
- MEDIAN: add smallint, int, bigint, and numeric variants
- Added support for DEFINE_COLUMN_LONG, COLUMN_VALUE_LONG and LAST_ERROR_POSITION additional function/procedure in DBMS_SQL Package.
- Support for "utl_http.end_of_body" exception
- Support for function to_timestamp_tz() in EPAS.
- PARALLEL/NOPARALLEL option for CREATE TABLE and INDEX.
- Create index syntax contains column name and number i.e. (col_name,1).
- Log the number of rows processed for bulk execution.
- Consistency across CSV and XML audit logs.
- Edbldr to support all connection parameters like psql and other clients.
- Support for function/procedure forward declaration inside package body.
- Add support for AES192 & AES256 in DBMS_CRYPTO package.
- Enhance Redwood compatible view.
- Allow creating a compound trigger having WHEN clause with NEW/OLD variables and STATEMENT level triggering events.
- Fix split partition behavior to be more like redwood.
- Enhancement around edbldr and multi-insert.
- More enhancement in EDB-SPL language.
- Support FM format in to_number() function.
- SYSDATE to behave more like Oracle (STABLE).

Summary

Postgres is getting faster and more capable

- TPS improvements of over 50% in 4 years
- Parallel vacuum of indexes: 4 X faster
- New and enhanced capabilities
  - Vacuum for append only tables
  - Security and consistency
  - Data Loading
  - Partitioning





Questions?

EDB

Thank You