EDB Postgres Advanced Server

Dilip Kumar; Database Architect

November 2022

EDB

Agenda

- Introduction
- Audit logging
- Optimizer Hints
- Automatic Partitioning
- Built in package support
- Autonomous transaction
- Tools Support
  - EDB wait state
  - EDB index advisor
  - pg_catcheck
- Distributed support

Notes: (Remove this)

1. Intro
   a. https://www.youtube.com/watch?v=Q9PD1xTzY1U
   b.
2. Security
   a. https://www.youtube.com/watch?v=NHKKuXarE90&list=PLownlFUq_rL7SNrGT5tO3TjZVbieN1vWQ&index=1
   b. https://www.youtube.com/watch?v=fq-LDdPSbEY

1) Partitioning 2) Redaction 3) edb_wait_state, index_advisor, SQL Protect (can be one slide).

2)Compound Trigger is another topic.

3) Auditing feature

4) Connect By

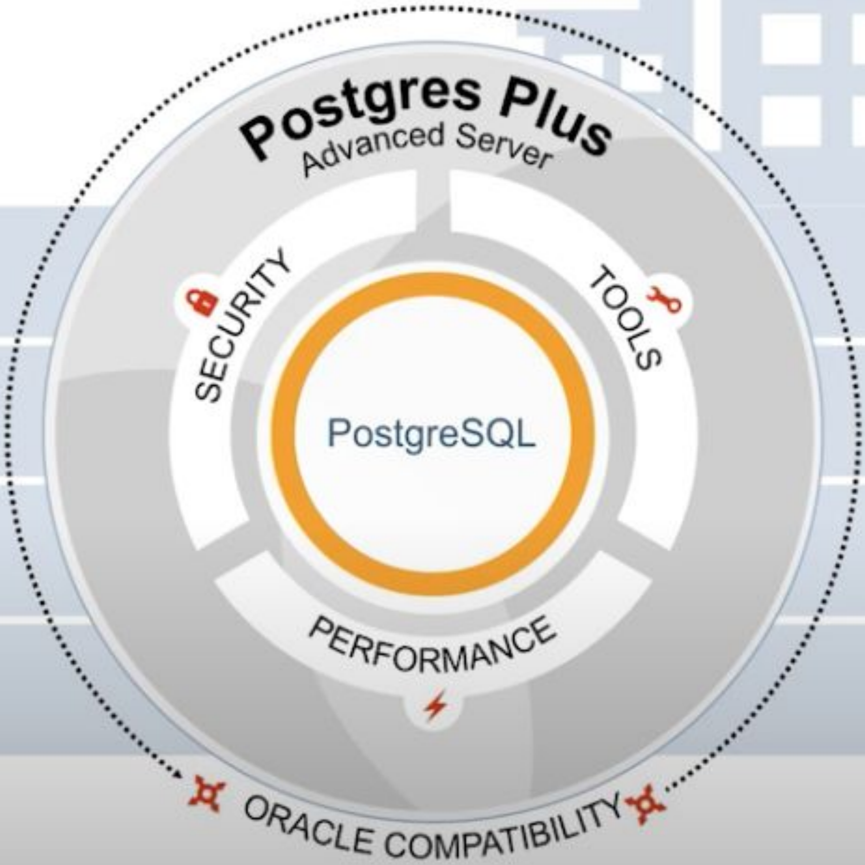
5) Optimizer Hints

6) Advance Queueing

POSTGRES PLUS

ADVANCED SERVER

- Security
- Tools
- Performance
- Compatibility



Audit Logging

- Database auditing allows database administrators, auditors, and operators to track and analyze database activities in support of complex auditing requirements.

- These audit activities include database access and usage along with data creation, change, or deletion.

- It’s recommended that you audit the following items:
    - User connections
    - DDL changes
    - Data changes
    - Data views

Audit Logging

- EDB Audit Logging generates audit log files, which contains all of the relevant information

- The audit logs can be configured to record information such as:
    - When a role establishes a connection to an EDB Postgres Advanced Server database
    - What database objects a role creates, modifies, or deletes when connected to EDB Postgres Advanced Server
    - When any failed authentication attempts occur

Audit Logging

Statement level audit logging

- Highly detailed levels of scrutiny can result in a lot of log messages—so log only at the level you need

- The `edb_audit_statement` permits inclusion of one or more, comma-separated values to control which SQL statements are to be audited.

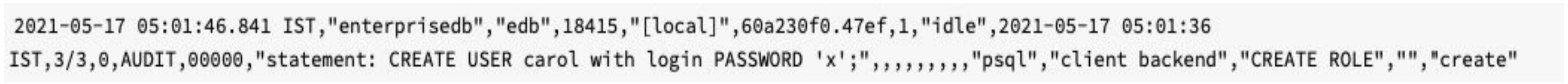
- Some example for these values could be all, ddl, dm, error, select, select | insert | update etc.

Audit Logging

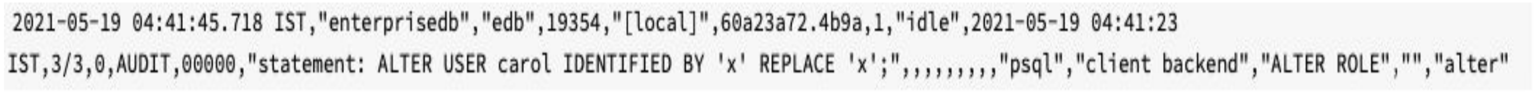
Redact password in audit log

- In open source PostgreSQL, a high logging level, combined with storage of passwords in the database, can result in passwords being displayed in the logs
- With EDB audit logging you can set the GUC parameter edb_filter_log.redact_password_commands to redact passwords in the logfile.

```
CREATE USER carol with login PASSWORD '1safepwd';
ALTER USER carol IDENTIFIED BY 'new_pass' REPLACE '1safepwd';
```



```
2021-05-17 05:01:46.841 IST,"enterprisedb","edb",18415,"[local]",60a230f0.47ef,1,"idle",2021-05-17 05:01:36
IST,3/3,0,AUDIT,00000,"statement: CREATE USER carol with login PASSWORD 'x';",,,,,,,,,"psql","client backend","CREATE ROLE","","create"
```



```
2021-05-19 04:41:45.718 IST,"enterprisedb","edb",19354,"[local]",60a23a72.4b9a,1,"idle",2021-05-19 04:41:23
IST,3/3,0,AUDIT,00000,"statement: ALTER USER carol IDENTIFIED BY 'x' REPLACE 'x';",,,,,,,,,"psql","client backend","ALTER ROLE","","alter"
```

Audit Logging

Object level auditing

- With Postgres, you can adjust logging levels on a per-user and per-database basis. In EDB Postgres Advanced Server, you can also control logging at the object type level, such as tables, views, triggers, function, etc.

- The object-level auditing allows selective auditing of objects for specific Data Manipulation Language Statements, such as `SELECT`, `UPDATE`, `DELETE`, and `INSERT` on a given table

- The object-level auditing also lets you include or exclude specific groups by specifying `(@)` or `(-)` with the `edb_audit_statement` parameter.

Partitioning

- Partitioning is one of the very important feature when it comes to distributing your data but it always come with the maintenance cost.
- Suppose you add new data which are not fitting into current partitions then you might need to manually add the partitions.
- But with Advance server we provide something called automatic partitioning

Partition Automation

- Automatically create a new partitions
- Easy way to manage the partition scheme
- Don’t need to worry about partition creation at runtime
- Easy LOAD and COPY data from old non-partiion to partition table
- Type of automatic partitioning
    - INTERVAL Partition (RANGE)
    - AUTOMATIC PArtition (LIST)
    - Provide partition and sub-partition number (HASH)

Interval Partitioning

- 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 Partitioning

```
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 Partitioning

```
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 Partitioning

- 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 Partitioning

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

- "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

```
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

- 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

```
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)
```

Autonomous Transactions

- An SPL program can be declared as an autonomous transaction
- An *autonomous transaction* is an independent transaction started by a calling program
- A commit or rollback of SQL commands within the autonomous transaction has no effect on the commit or rollback in any transaction of the calling program
- A commit or rollback in the calling program has no effect on the commit or rollback of SQL commands in the autonomous transaction.

Packages

- *H*

Stored Procedure

-

1) Redwood compatible Partition Syntax
2) Auto Partitioning
3) Template

Last DDL Time

Connect By

EDB wait state

Index Advisor

- The Index Advisor utility helps determine which columns you should index to improve performance in a given workload.
- It collect the qual information from real time queries
- Generate the hypothetical indexes based on qual and check which indexes are actually used by the optimizer
- Suggest indexes based on its usability by planner with actual queries

Postgres Distributed

- DDL automatically
- Conflict Resolution automatically
- Multi Master Replication
- CAMO
- Global Sequences
- Always On

(no global locking except DDL and Global Sequences)

Impact of clock synchronisation

Postgres-BDR

Automatic DDL and DML replication in an active-active mesh network

Failover and switchover infrastructure to re-route traffic in case of failures or during maintenance operation

Advanced conflict detection and conflict management

Management of distributed sequences

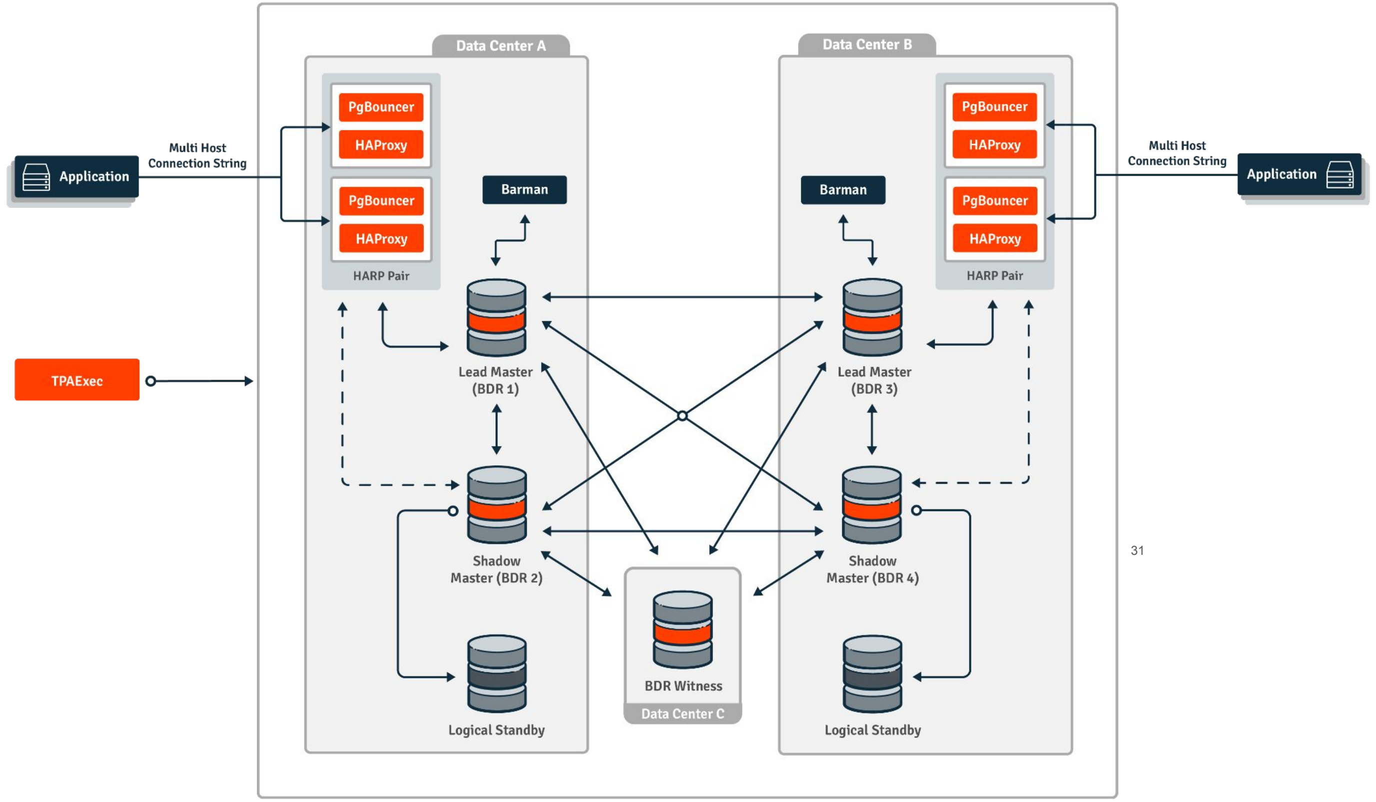
Differentiated replication sets to control which data gets replicated and to which downstream databases

Cluster expansion/consolidation

Rolling database software upgrades

Rolling schema change/migration using cross-schema replication

Postgres-BDR - Always On Platinum


Data Center A
Data Center B
PgBouncer
HAProxy
PgBouncer
HAProxy
HARP Pair
Barman
Application
Multi Host Connection String
TPAExec
Lead Master (BDR 1)
Shadow Master (BDR 2)
Logical Standby
Lead Master (BDR 3)
Shadow Master (BDR 4)
BDR Witness
Data Center C