



EDB

EDB Postgres Advanced Server Guide

Release 15.2.0

EnterpriseDB Corporation

Feb 17, 2023

CONTENTS

1	Introduction	3
1.1	Hard limits	3
1.1.1	Conventions used	4
1.1.2	About the examples	4
2	Enhanced compatibility features	15
2.1	Enabling compatibility features	15
2.2	Stored procedural language	15
2.3	Optimizer hints	16
2.4	Data dictionary views	16
2.5	dblink_ora	16
2.6	Profile management	16
2.7	Built-in packages	17
2.8	Open Client Library	18
2.9	Utilities	18
2.9.1	EDB*Plus	18
2.9.2	EDB*Loader	20
2.9.3	EDB*Wrap	20
2.9.4	Dynamic Runtime Instrumentation Tools Architecture (DRITA)	20
2.10	ECPGPlus	20
2.11	Table partitioning	21
3	Database administration	23
3.1	Configuration parameters	23
3.1.1	Setting configuration parameters	23
3.1.2	Summary of configuration parameters	25
3.1.3	Configuration parameters by functionality	47
3.2	Index Advisor	86
3.2.1	Index Advisor components	87
3.2.2	Index Advisor configuration	87
3.2.3	Using Index Advisor	89
3.2.4	Reviewing the Index Advisor recommendations	92
3.2.5	Index Advisor limitations	96
3.3	SQL Profiler	96
3.3.1	Troubleshooting	98
3.4	pgsnmpd	98
3.4.1	Configuring pgsnmpd	98
3.4.2	Setting the listener address	99
3.4.3	Invoking pgsnmpd	99
3.4.4	Viewing pgsnmpd help	100

3.4.5	Requesting information from pgsnmpd	100
3.5	EDB audit logging	100
3.5.1	Audit logging configuration parameters	101
3.5.2	Selecting SQL statements to audit	103
3.5.3	Enabling audit logging	116
3.5.4	Audit log file	121
3.5.5	Using error codes to filter audit logs	127
3.5.6	Using command tags to filter audit logs	128
3.5.7	Redacting passwords in audit logs	129
3.5.8	Audit log archiving	130
3.5.9	Object auditing	133
3.6	Unicode collation algorithm	138
3.6.1	Basic Unicode collation algorithm concepts	139
3.6.2	International components for Unicode	140
3.6.3	Using a collation	142
4	EDB Resource Manager	147
4.1	Creating and managing resource groups	148
4.1.1	CREATE RESOURCE GROUP	148
4.1.2	Description	148
4.1.3	Parameters	148
4.1.4	Example	148
4.1.5	ALTER RESOURCE GROUP	149
4.1.6	Description	149
4.1.7	Parameters	149
4.1.8	Example	150
4.1.9	DROP RESOURCE GROUP	150
4.1.10	Description	150
4.1.11	Parameters	150
4.1.12	Example	151
4.1.13	Assigning a process to a resource group	151
4.1.14	Removing a process from a resource group	152
4.1.15	Monitoring processes in resource groups	152
4.2	CPU usage throttling	153
4.2.1	Setting the CPU rate limit for a resource group	154
4.2.2	Example: Single process in a single group	155
4.2.3	Example: Multiple processes in a single group	156
4.2.4	Example: Multiple processes in multiple groups	157
4.3	Dirty buffer throttling	160
4.3.1	Setting the dirty rate limit for a resource group	160
4.3.2	Example: Single process in a single group	161
4.3.3	Example: Multiple processes in a single group	163
4.3.4	Example: Multiple processes in multiple groups	165
4.4	System catalogs	168
4.4.1	edb_all_resource_groups	168
4.4.2	edb_resource_group	168
5	libpq C library	169
5.1	Using libpq with EDB SPL	169
5.2	REFCURSOR support	169
5.3	Returning a single REFCURSOR	170
5.3.1	Returning multiple REFCURSORs	172
5.4	Array binding	175
5.4.1	PQBulkStart	175

5.4.2	API definition	175
5.4.3	PQexecBulk	176
5.4.4	API definition	176
5.4.5	PQBulkFinish	176
5.4.6	API definition	176
5.4.7	PQexecBulkPrepared	176
5.4.8	API definition	176
5.4.9	Example code (Using PQBulkStart, PQexecBulk, PQBulkFinish)	177
5.4.10	Example code (Using PQexecBulkPrepared)	178
6	Debugger	179
6.1	Configuring the Debugger	180
6.2	Starting the Debugger	180
6.3	The Debugger window	181
6.4	Main Debugger window	183
6.4.1	The Program Body panel	183
6.4.2	The Tabs panel	184
6.4.3	The Stack tab	187
6.5	Debugging a program	189
6.5.1	Stepping through the code	189
6.5.2	Using breakpoints	189
6.5.3	Removing a local breakpoint	190
6.5.4	Setting a global breakpoint for in-context debugging	191
6.5.5	Exiting the Debugger	194
7	Performance analysis and tuning	195
7.1	Dynatune	195
7.1.1	edb_dynatune	195
7.1.2	edb_dynatune_profile	196
7.2	EDB wait states	196
7.2.1	EDB Wait States Background Worker (EWSBW)	196
7.2.2	Terminating the wait states worker	197
7.2.3	The wait states interface	197
7.2.4	edb_wait_states_data	198
7.2.5	Parameters	199
7.2.6	Example	199
7.2.7	edb_wait_states_queries	200
7.2.8	Parameters	200
7.2.9	Example	201
7.2.10	edb_wait_states_sessions	201
7.2.11	Parameters	202
7.2.12	Example	202
7.2.13	edb_wait_states_samples	202
7.2.14	Parameters	203
7.2.15	Example	203
7.2.16	edb_wait_states_purge	204
7.2.17	Example	204
8	EDB clone schema	207
8.1	Setup process	209
8.1.1	Installing extensions and PL/Perl	209
8.1.2	Setting configuration parameters	210
8.1.3	Performance configuration parameters	210
8.1.4	Status logging	211

8.1.5	Installing EDB Clone Schema	211
8.1.6	Creating the foreign servers and user mappings	212
8.1.7	Foreign server and user mapping for local cloning functions	212
8.1.8	Foreign server and user mapping for remote cloning functions	213
8.2	EDB Clone Schema functions	215
8.2.1	localcopyschema	216
8.2.2	Parameters	216
8.2.3	Example	217
8.2.4	localcopyschema_nb	219
8.2.5	Parameters	220
8.2.6	Example	221
8.2.7	remotecopyschema	222
8.2.8	Parameters	222
8.2.9	Example	223
8.2.10	remotecopyschema_nb	226
8.2.11	Parameters	226
8.2.12	Example	227
8.2.13	process_status_from_log	228
8.2.14	Parameters	228
8.2.15	Example	228
8.2.16	remove_log_file_and_job	229
8.2.17	Parameters	229
8.2.18	Example	229
9	Enhanced SQL and other miscellaneous features	231
9.1	COMMENT	231
9.1.1	Parameters	232
9.1.2	Notes	234
9.1.3	Example	234
9.2	Output of function version()	234
9.3	Logical decoding on standby	235
10	System catalog tables	237
10.1	edb_dir	237
10.2	edb_all_resource_groups	237
10.3	edb_policy	238
10.4	edb_profile	239
10.5	edb_redaction_column	240
10.6	edb_redaction_policy	240
10.7	edb_resource_group	241
10.8	edb_variable	241
10.9	pg_synonym	241
10.10	product_component_version	242
11	EDB Postgres Advanced Server exceptions	243
11.1	EDB Postgres Advanced Server keywords	244

EDB Postgres Advanced Server adds extended functionality to the open-source PostgreSQL database. The extended functionality supports database administration, enhanced SQL capabilities, database and application security, performance monitoring and analysis, and application development utilities. This guide documents those features that are exclusive to EDB Postgres Advanced Server.

Enhanced compatibility features. This section provides an overview of compatibility features supported by EDB Postgres Advanced Server.

Database administration. This section contains information about features and tools that are helpful to the database administrator.

- **Index Advisor** helps to determine the additional indexes needed on tables to improve application performance.
- **SQL Profiler** locates and diagnoses poorly running SQL queries in applications.
- **pgsnmpd** is an SNMP agent that returns hierarchical monitoring information regarding the current state of EDB Postgres Advanced Server.

EDB Resource Manager. This section contains information about the EDB Resource Manager feature, which provides the capability to control system resource usage by EDB Postgres Advanced Server processes.

- **Resource Groups** shows how to create and maintain the groups on which resource limits can be defined.
- **CPU Usage Throttling** provides a method to control CPU usage by EDB Postgres Advanced Server processes.
- **Dirty Buffer Throttling** provides a method to control the dirty rate of shared buffers by EDB Postgres Advanced Server processes.

The libpq C library. The libpq C library is the C application programming interface (API) language for EDB Postgres Advanced Server.

The PL Debugger. The PL Debugger is a graphically oriented debugging tool for PL/pgSQL.

Performance analysis and tuning. This section contains the various tools for analyzing and improving application and database server performance.

- **Dynatune** provides a quick and easy means for configuring EDB Postgres Advanced Server depending upon the type of application usage.
- **EDB wait states** provides a way to capture wait events and other data for performance diagnosis.

EDB Clone Schema. This section contains information about the EDB Clone Schema feature, which provides the capability to copy a schema and its database objects within a single database or from one database to another database.

Enhanced SQL and other miscellaneous features. This section contains information on enhanced SQL functionality and other features that provide additional flexibility and convenience.

System catalog tables. This section contains additional *system catalog tables* added for EDB Postgres Advanced Server specific database objects.

EDB Postgres Advanced Server exceptions. This section contains information about the predefined exceptions, the SQLstate values, associated redwood error code, and description of the exceptions.

EDB Postgres Advanced Server keywords. This section contains information about the words that EDB Postgres Advanced Server recognizes as keywords.

For information about the features that are shared by EDB Postgres Advanced Server and PostgreSQL, see the [PostgreSQL core documentation](#).

INTRODUCTION

See the release notes for the features added in EDB Postgres Advanced Server 14.

1.1 Hard limits

The following table describes various hard limits of PostgreSQL. However, practical limits such as performance limitations or available disk space might apply before absolute hard limits are reached.

Item	Upper limit	Comment
<i>database size</i>	unlimited	
<i>number of databases</i>	4,294,950,911	
<i>relations per database</i>	1,431,650,303	
<i>relation size</i>	32 TB	with the default BLCKSZ of 8192 bytes
<i>rows per table</i>	limited by the number of tuples that can fit onto 4,294,967,295 pages	
<i>columns per table</i>	1600	further limited by tuple size fitting on a single page; see note below
<i>field size</i>	1 GB	
<i>identifier length</i>	63 bytes	can be increased by recompiling PostgreSQL
<i>indexes per table</i>	unlimited	constrained by maximum relations per database
<i>columns per index</i>	32	can be increased by recompiling PostgreSQL
<i>partition keys</i>	32	can be increased by recompiling PostgreSQL

Note:

- The maximum number of columns for a table is further reduced as the tuple being stored must fit in a single 8192-byte heap page. For example, excluding the tuple header, a tuple made up of 1600 *int* columns consumes 6400 bytes and can be stored in a heap page. But a tuple of 1600 *bigint* columns consumes 12800 bytes and therefore doesn't fit inside a heap page. Variable-length fields of types such as *text*, *varchar*, and *char* can have their values stored out of line in the table's *TOAST* table when the values are large enough to require it. Only an 18-byte pointer must remain inside the tuple in the table's heap. For shorter length variable-length fields, either a 4-byte or 1-byte field header is used, and the value is stored inside the heap tuple.

- Columns that were dropped **from the** table also contribute to the maximum column limit.
- Moreover, although the dropped column values **for** newly created tuples are internally
- marked **as null in** the **tuple's null bitmap**, the **null bitmap also occupies space**.

1.1.1 Conventions used

This information applies to both Linux and Windows systems. Directory paths are presented in the Linux format with forward slashes. When working on Windows systems, start the directory path with the drive letter followed by a colon and substitute back slashes for forward slashes.

Some of this information might apply interchangeably to the PostgreSQL and EDB Postgres Advanced Server database systems. The term *EDB Postgres Advanced Server* is used to refer to EDB Postgres Advanced Server. The term *Postgres* is used to generically refer to both PostgreSQL and EDB Postgres Advanced Server. When a distinction needs to be made between these two database systems, the specific names—PostgreSQL or EDB Postgres Advanced Server—are used.

The installation directory path of the PostgreSQL or EDB Postgres Advanced Server products is referred to as *PGRES_INSTALL_HOME*.

- For PostgreSQL Linux installations, the default is `/opt/PostgreSQL/x.x` for version 10 and earlier. For later versions, use the PostgreSQL community packages.
- For EDB Postgres Advanced Server Linux installations performed using the interactive installer for version 10 and earlier, the default is `/opt/edb/asx.x`.
- For EDB Postgres Advanced Server Linux installations performed using an RPM package, the default is `/usr/edb/asxx`.
- For EDB Postgres Advanced Server Windows installations, the default is `C:\Program Files\edb\asxx`. The product version number is represented by `x.x` or by `xx` for version 10 and later.

1.1.2 About the examples

The examples use the sample tables `dept`, `emp`, and `jobhist`, which are created and loaded when EDB Postgres Advanced Server is installed.

You can re-create the tables and programs in the sample database at any time by executing the following script:

```
/usr/edb/asxx/share/pg-sample.sql
```

Where `xx` is the EDB Postgres Advanced Server version number.

In addition, a script in the same directory contains the database objects created using syntax compatible with Oracle databases. This script file is `edb-sample.sql`. The script:

- Creates the sample tables and programs in the currently connected database.
- Grants all permissions on the tables to the `PUBLIC` group.

The tables and programs are created in the first schema of the search path in which the current user has permission to create tables and procedures. You can display the search path using the command:

```
SHOW SEARCH_PATH;
```

You can use `PSQL` commands to modify the search path.

Sample database description

The sample database represents employees in an organization. It contains three types of records: employees, departments, and historical records of employees.

Each employee has an identification number, name, hire date, salary, and manager. Some employees earn a commission in addition to their salary. All employee-related information is stored in the `emp` table.

The sample company is regionally diverse, so it tracks the locations of its departments. Each company employee is assigned to a department. Each department is identified by a unique department number and a short name. Each department is associated with one location. All department-related information is stored in the `dept` table.

The company also tracks information about jobs held by the employees. Some employees have been with the company for a long time and have held different positions, received raises, switched departments, and so on. When a change in employee status occurs, the company records the end date of the former position. A new job record is added with the start date and the new job title, department, salary, and the reason for the status change. All employee history is maintained in the `jobhist` table.

The following is the `pg-sample.sql` script:

```
SET datestyle TO iso, dmy;

--
-- Script that creates the sample tables, views
-- functions, triggers, etc.
--
-- Start new transaction - commit all or nothing
--
BEGIN;
--
-- Create and load tables used in the documentation examples.
--
-- Create the dept table
--
CREATE TABLE dept (
    deptno      NUMERIC(2) NOT NULL CONSTRAINT dept_pk PRIMARY KEY,
    dname       VARCHAR(14) CONSTRAINT dept_dname_uq UNIQUE,
    loc         VARCHAR(13)
);
--
-- Create the emp table
--
CREATE TABLE emp (
    empno       NUMERIC(4) NOT NULL CONSTRAINT emp_pk PRIMARY KEY,
    ename       VARCHAR(10),
    job         VARCHAR(9),
    mgr         NUMERIC(4),
    hiredate    DATE,
    sal         NUMERIC(7,2) CONSTRAINT emp_sal_ck CHECK (sal > 0),
    comm        NUMERIC(7,2),
    deptno      NUMERIC(2) CONSTRAINT emp_ref_dept_fk
                REFERENCES dept(deptno)
);
--
-- Create the jobhist table
```

(continues on next page)

(continued from previous page)

```

- -
CREATE TABLE jobhist (
    empno          NUMERIC(4) NOT NULL,
    startdate      TIMESTAMP(0) NOT NULL,
    enddate        TIMESTAMP(0),
    job            VARCHAR(9),
    sal            NUMERIC(7,2),
    comm           NUMERIC(7,2),
    deptno         NUMERIC(2),
    chgdesc        VARCHAR(80),
    CONSTRAINT jobhist_pk PRIMARY KEY (empno, startdate),
    CONSTRAINT jobhist_ref_emp_fk FOREIGN KEY (empno)
        REFERENCES emp(empno) ON DELETE CASCADE,
    CONSTRAINT jobhist_ref_dept_fk FOREIGN KEY (deptno)
        REFERENCES dept (deptno) ON DELETE SET NULL,
    CONSTRAINT jobhist_date_chk CHECK (startdate <= enddate)
);
- -
- - Create the salesemp view
- -
CREATE OR REPLACE VIEW salesemp AS
    SELECT empno, ename, hiredate, sal, comm FROM emp WHERE job = SALESMAN;
- -
- - Sequence to generate values for function new_empno.
- -
CREATE SEQUENCE next_empno START WITH 8000 INCREMENT BY 1;
- -
- - Issue PUBLIC grants
- -
-GRANT ALL ON emp TO PUBLIC;
-GRANT ALL ON dept TO PUBLIC;
-GRANT ALL ON jobhist TO PUBLIC;
-GRANT ALL ON salesemp TO PUBLIC;
-GRANT ALL ON next_empno TO PUBLIC;
- -
- - Load the dept table
- -
INSERT INTO dept VALUES (10,ACCOUNTING,NEW YORK);
INSERT INTO dept VALUES (20,RESEARCH,DALLAS);
INSERT INTO dept VALUES (30,SALES,CHICAGO);
INSERT INTO dept VALUES (40,OPERATIONS,BOSTON);
- -
- - Load the emp table
- -
INSERT INTO emp VALUES (7369,SMITH,CLERK,7902,17-DEC-80,800,NULL,20);
INSERT INTO emp VALUES (7499,ALLEN,SALESMAN,7698,20-FEB-81,1600,300,30);
INSERT INTO emp VALUES (7521,WARD,SALESMAN,7698,22-FEB-81,1250,500,30);
INSERT INTO emp VALUES (7566,JONES,MANAGER,7839,02-APR-81,2975,NULL,20);
INSERT INTO emp VALUES (7654,MARTIN,SALESMAN,7698,28-SEP-81,1250,1400,30);
INSERT INTO emp VALUES (7698,BLAKE,MANAGER,7839,01-MAY-81,2850,NULL,30);
INSERT INTO emp VALUES (7782,CLARK,MANAGER,7839,09-JUN-81,2450,NULL,10);
INSERT INTO emp VALUES (7788,SCOTT,ANALYST,7566,19-APR-87,3000,NULL,20);

```

(continues on next page)

(continued from previous page)

```

INSERT INTO emp VALUES (7839,KING,PRESIDENT,NULL,17-NOV-81,5000,NULL,10);
INSERT INTO emp VALUES (7844,TURNER,SALESMAN,7698,08-SEP-81,1500,0,30);
INSERT INTO emp VALUES (7876,ADAMS,CLERK,7788,23-MAY-87,1100,NULL,20);
INSERT INTO emp VALUES (7900,JAMES,CLERK,7698,03-DEC-81,950,NULL,30);
INSERT INTO emp VALUES (7902,FORD,ANALYST,7566,03-DEC-81,3000,NULL,20);
INSERT INTO emp VALUES (7934,MILLER,CLERK,7782,23-JAN-82,1300,NULL,10);
- -
- - Load the jobhist table
- -
INSERT INTO jobhist VALUES (7369,17-DEC-80,NULL,CLERK,800,NULL,20,New Hire);
INSERT INTO jobhist VALUES (7499,20-FEB-81,NULL,SALESMAN,1600,300,30,New Hire);
INSERT INTO jobhist VALUES (7521,22-FEB-81,NULL,SALESMAN,1250,500,30,New Hire);
INSERT INTO jobhist VALUES (7566,02-APR-81,NULL,MANAGER,2975,NULL,20,New Hire);
INSERT INTO jobhist VALUES (7654,28-SEP-81,NULL,SALESMAN,1250,1400,30,New Hire);
INSERT INTO jobhist VALUES (7698,01-MAY-81,NULL,MANAGER,2850,NULL,30,New Hire);
INSERT INTO jobhist VALUES (7782,09-JUN-81,NULL,MANAGER,2450,NULL,10,New Hire);
INSERT INTO jobhist VALUES (7788,19-APR-87,12-APR-88,CLERK,1000,NULL,20,New Hire);
INSERT INTO jobhist VALUES (7788,13-APR-88,04-MAY-89,CLERK,1040,NULL,20,Raise);
INSERT INTO jobhist VALUES (7788,05-MAY-90,NULL,ANALYST,3000,NULL,20,Promoted to_
↪Analyst);
INSERT INTO jobhist VALUES (7839,17-NOV-81,NULL,PRESIDENT,5000,NULL,10,New Hire);
INSERT INTO jobhist VALUES (7844,08-SEP-81,NULL,SALESMAN,1500,0,30,New Hire);
INSERT INTO jobhist VALUES (7876,23-MAY-87,NULL,CLERK,1100,NULL,20,New Hire);
INSERT INTO jobhist VALUES (7900,03-DEC-81,14-JAN-83,CLERK,950,NULL,10,New Hire);
INSERT INTO jobhist VALUES (7900,15-JAN-83,NULL,CLERK,950,NULL,30,Changed to Dept 30);
INSERT INTO jobhist VALUES (7902,03-DEC-81,NULL,ANALYST,3000,NULL,20,New Hire);
INSERT INTO jobhist VALUES (7934,23-JAN-82,NULL,CLERK,1300,NULL,10,New Hire);
- -
- - Populate statistics table and view (pg_statistic/pg_stats)
- -
ANALYZE dept;
ANALYZE emp;
ANALYZE jobhist;
- -
- - Function that lists all employees numbers and names
- - from the emp table using a cursor.
- -
CREATE OR REPLACE FUNCTION list_emp() RETURNS VOID
AS $$
DECLARE
    v_empno      NUMERIC(4);
    v_ename      VARCHAR(10);
    emp_cur CURSOR FOR
        SELECT empno, ename FROM emp ORDER BY empno;
BEGIN
    OPEN emp_cur;
    RAISE INFO EMPNO ENAME;
    RAISE INFO -----;
    LOOP
        FETCH emp_cur INTO v_empno, v_ename;
        EXIT WHEN NOT FOUND;
        RAISE INFO %, v_empno, v_ename;
    
```

(continues on next page)

(continued from previous page)

```

    END LOOP;
    CLOSE emp_cur;
    RETURN;
END;
$$ LANGUAGE plpgsql;
--
-- Function that selects an employee row given the employee
-- number and displays certain columns.
--
CREATE OR REPLACE FUNCTION select_emp (
    p_empno          NUMERIC
) RETURNS VOID
AS $$
DECLARE
    v_ename          emp.ename%TYPE;
    v_hiredate        emp.hiredate%TYPE;
    v_sal             emp.sal%TYPE;
    v_comm            emp.comm%TYPE;
    v_dname           dept.dname%TYPE;
    v_disp_date       VARCHAR(10);
BEGIN
    SELECT INTO
        v_ename, v_hiredate, v_sal, v_comm, v_dname
        ename, hiredate, sal, COALESCE(comm, 0), dname
    FROM emp e, dept d
    WHERE empno = p_empno
        AND e.deptno = d.deptno;
    IF NOT FOUND THEN
        RAISE INFO Employee % not found, p_empno;
        RETURN;
    END IF;
    v_disp_date := TO_CHAR(v_hiredate, MM/DD/YYYY);
    RAISE INFO Number      : %, p_empno;
    RAISE INFO Name        : %, v_ename;
    RAISE INFO Hire Date   : %, v_disp_date;
    RAISE INFO Salary      : %, v_sal;
    RAISE INFO Commission: %, v_comm;
    RAISE INFO Department: %, v_dname;
    RETURN;
EXCEPTION
    WHEN OTHERS THEN
        RAISE INFO The following is SQLERRM : %, SQLERRM;
        RAISE INFO The following is SQLSTATE: %, SQLSTATE;
        RETURN;
END;
$$ LANGUAGE plpgsql;
--
-- A RECORD type used to format the return value of
-- function, emp_query.
--
CREATE TYPE emp_query_type AS (
    empno          NUMERIC,

```

(continues on next page)

(continued from previous page)

```

    ename          VARCHAR(10),
    job            VARCHAR(9),
    hiredate       DATE,
    sal            NUMERIC
);
- -
- - Function that queries the emp table based on
- - department number and employee number or name. Returns
- - employee number and name as INOUT parameters and job,
- - hire date, and salary as OUT parameters. These are
- - returned in the form of a record defined by
- - RECORD type, emp_query_type.
- -
CREATE OR REPLACE FUNCTION emp_query (
    IN p_deptno      NUMERIC,
    INOUT p_empno     NUMERIC,
    INOUT p_ename     VARCHAR,
    OUT p_job         VARCHAR,
    OUT p_hiredate    DATE,
    OUT p_sal         NUMERIC
)
AS $$
BEGIN
    SELECT INTO
        p_empno, p_ename, p_job, p_hiredate, p_sal
        empno, ename, job, hiredate, sal
    FROM emp
    WHERE deptno = p_deptno
        AND (empno = p_empno
            OR ename = UPPER(p_ename));
END;
$$ LANGUAGE plpgsql;
- -
- - Function to call emp_query_caller with IN and INOUT
- - parameters. Displays the results received from INOUT and
- - OUT parameters.
- -
CREATE OR REPLACE FUNCTION emp_query_caller() RETURNS VOID
AS $$
DECLARE
    v_deptno        NUMERIC;
    v_empno          NUMERIC;
    v_ename          VARCHAR;
    v_rows           INTEGER;
    r_emp_query      EMP_QUERY_TYPE;
BEGIN
    v_deptno := 30;
    v_empno := 0;
    v_ename := Martin;
    r_emp_query := emp_query(v_deptno, v_empno, v_ename);
    RAISE INFO Department : %, v_deptno;
    RAISE INFO Employee No: %, (r_emp_query).empno;

```

(continues on next page)

(continued from previous page)

```

        RAISE INFO Name      : %, (r_emp_query).ename;
        RAISE INFO Job       : %, (r_emp_query).job;
        RAISE INFO Hire Date : %, (r_emp_query).hiredate;
        RAISE INFO Salary    : %, (r_emp_query).sal;
        RETURN;
    EXCEPTION
        WHEN OTHERS THEN
            RAISE INFO The following is SQLERRM : %, SQLERRM;
            RAISE INFO The following is SQLSTATE: %, SQLSTATE;
            RETURN;
END;
$$ LANGUAGE plpgsql;
--
-- Function to compute yearly compensation based on semimonthly
-- salary.
--
CREATE OR REPLACE FUNCTION emp_comp (
    p_sal      NUMERIC,
    p_comm     NUMERIC
) RETURNS NUMERIC
AS $$
BEGIN
    RETURN (p_sal + COALESCE(p_comm, 0)) * 24;
END;
$$ LANGUAGE plpgsql;
--
-- Function that gets the next number from sequence, next_empno,
-- and ensures it is not already in use as an employee number.
--
CREATE OR REPLACE FUNCTION new_empno() RETURNS INTEGER
AS $$
DECLARE
    v_cnt      INTEGER := 1;
    v_new_empno INTEGER;
BEGIN
    WHILE v_cnt > 0 LOOP
        SELECT INTO v_new_empno nextval(next_empno);
        SELECT INTO v_cnt COUNT(*) FROM emp WHERE empno = v_new_empno;
    END LOOP;
    RETURN v_new_empno;
END;
$$ LANGUAGE plpgsql;
--
-- Function that adds a new clerk to table emp.
--
CREATE OR REPLACE FUNCTION hire_clerk (
    p_ename    VARCHAR,
    p_deptno   NUMERIC
) RETURNS NUMERIC
AS $$
DECLARE
    v_empno    NUMERIC(4);

```

(continues on next page)

(continued from previous page)

```

v_ename      VARCHAR(10);
v_job        VARCHAR(9);
v_mgr        NUMERIC(4);
v_hiredate   DATE;
v_sal        NUMERIC(7,2);
v_comm       NUMERIC(7,2);
v_deptno     NUMERIC(2);
BEGIN
    v_empno := new_empno();
    INSERT INTO emp VALUES (v_empno, p_ename, CLERK, 7782,
        CURRENT_DATE, 950.00, NULL, p_deptno);
    SELECT INTO
        v_empno, v_ename, v_job, v_mgr, v_hiredate, v_sal, v_comm, v_deptno
        empno, ename, job, mgr, hiredate, sal, comm, deptno
        FROM emp WHERE empno = v_empno;
    RAISE INFO Department : %, v_deptno;
    RAISE INFO Employee No: %, v_empno;
    RAISE INFO Name       : %, v_ename;
    RAISE INFO Job        : %, v_job;
    RAISE INFO Manager    : %, v_mgr;
    RAISE INFO Hire Date  : %, v_hiredate;
    RAISE INFO Salary     : %, v_sal;
    RAISE INFO Commission : %, v_comm;
    RETURN v_empno;
EXCEPTION
    WHEN OTHERS THEN
        RAISE INFO The following is SQLERRM : %, SQLERRM;
        RAISE INFO The following is SQLSTATE: %, SQLSTATE;
        RETURN -1;
END;
$$ LANGUAGE plpgsql;
--
-- Function that adds a new salesman to table emp.
--
CREATE OR REPLACE FUNCTION hire_salesman (
    p_ename      VARCHAR,
    p_sal        NUMERIC,
    p_comm       NUMERIC
) RETURNS NUMERIC
AS $$
DECLARE
    v_empno      NUMERIC(4);
    v_ename      VARCHAR(10);
    v_job        VARCHAR(9);
    v_mgr        NUMERIC(4);
    v_hiredate   DATE;
    v_sal        NUMERIC(7,2);
    v_comm       NUMERIC(7,2);
    v_deptno     NUMERIC(2);
BEGIN
    v_empno := new_empno();
    INSERT INTO emp VALUES (v_empno, p_ename, SALESMAN, 7698,

```

(continues on next page)

(continued from previous page)

```

        CURRENT_DATE, p_sal, p_comm, 30);
SELECT INTO
    v_empno, v_ename, v_job, v_mgr, v_hiredate, v_sal, v_comm, v_deptno
    empno, ename, job, mgr, hiredate, sal, comm, deptno
    FROM emp WHERE empno = v_empno;
RAISE INFO Department : %, v_deptno;
RAISE INFO Employee No: %, v_empno;
RAISE INFO Name       : %, v_ename;
RAISE INFO Job        : %, v_job;
RAISE INFO Manager    : %, v_mgr;
RAISE INFO Hire Date  : %, v_hiredate;
RAISE INFO Salary     : %, v_sal;
RAISE INFO Commission : %, v_comm;
RETURN v_empno;
EXCEPTION
    WHEN OTHERS THEN
        RAISE INFO The following is SQLERRM : %, SQLERRM;
        RAISE INFO The following is SQLSTATE: %, SQLSTATE;
        RETURN -1;
END;
$$ LANGUAGE plpgsql;
--
-- Rule to INSERT into view salesemp
--
CREATE OR REPLACE RULE salesemp_i AS ON INSERT TO salesemp
DO INSTEAD
    INSERT INTO emp VALUES (NEW.empno, NEW.ename, SALESMAN, 7698,
        NEW.hiredate, NEW.sal, NEW.comm, 30);
--
-- Rule to UPDATE view salesemp
--
CREATE OR REPLACE RULE salesemp_u AS ON UPDATE TO salesemp
DO INSTEAD
    UPDATE emp SET empno    = NEW.empno,
                  ename     = NEW.ename,
                  hiredate  = NEW.hiredate,
                  sal       = NEW.sal,
                  comm      = NEW.comm
    WHERE empno = OLD.empno;
--
-- Rule to DELETE from view salesemp
--
CREATE OR REPLACE RULE salesemp_d AS ON DELETE TO salesemp
DO INSTEAD
    DELETE FROM emp WHERE empno = OLD.empno;
--
-- After statement-level trigger that displays a message after
-- an insert, update, or deletion to the emp table. One message
-- per SQL command is displayed.
--
CREATE OR REPLACE FUNCTION user_audit_trig() RETURNS TRIGGER
AS $$

```

(continues on next page)

(continued from previous page)

```

DECLARE
    v_action          VARCHAR(24);
    v_text            TEXT;
BEGIN
    IF TG_OP = INSERT THEN
        v_action := added employee(s) on ;
    ELSIF TG_OP = UPDATE THEN
        v_action := updated employee(s) on ;
    ELSIF TG_OP = DELETE THEN
        v_action := deleted employee(s) on ;
    END IF;
    v_text := User || USER || v_action || CURRENT_DATE;
    RAISE INFO %, v_text;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER user_audit_trig
    AFTER INSERT OR UPDATE OR DELETE ON emp
    FOR EACH STATEMENT EXECUTE PROCEDURE user_audit_trig();
--
-- Before row-level trigger that displays employee number and
-- salary of an employee that is about to be added, updated,
-- or deleted in the emp table.
--
CREATE OR REPLACE FUNCTION emp_sal_trig() RETURNS TRIGGER
AS $$
DECLARE
    sal_diff          NUMERIC(7,2);
BEGIN
    IF TG_OP = INSERT THEN
        RAISE INFO Inserting employee %, NEW.empno;
        RAISE INFO ..New salary: %, NEW.sal;
        RETURN NEW;
    END IF;
    IF TG_OP = UPDATE THEN
        sal_diff := NEW.sal - OLD.sal;
        RAISE INFO Updating employee %, OLD.empno;
        RAISE INFO ..Old salary: %, OLD.sal;
        RAISE INFO ..New salary: %, NEW.sal;
        RAISE INFO ..Raise : %, sal_diff;
        RETURN NEW;
    END IF;
    IF TG_OP = DELETE THEN
        RAISE INFO Deleting employee %, OLD.empno;
        RAISE INFO ..Old salary: %, OLD.sal;
        RETURN OLD;
    END IF;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER emp_sal_trig
    BEFORE DELETE OR INSERT OR UPDATE ON emp
    FOR EACH ROW EXECUTE PROCEDURE emp_sal_trig();

```

(continues on next page)

(continued from previous page)

<code>COMMIT;</code>

ENHANCED COMPATIBILITY FEATURES

EDB Postgres Advanced Server includes extended functionality that provides compatibility for syntax supported by Oracle applications. See

Database Compatibility for Oracle Developers for detailed information about the compatibility features supported by EDB Postgres Advanced Server.

The following sections highlight some of the compatibility features supported by EDB Postgres Advanced Server.

2.1 Enabling compatibility features

There are several ways to install EDB Postgres Advanced Server that allow you to take advantage of compatibility features:

- Use the `INITDBOPTS` variable (in the EDB Postgres Advanced Server service configuration file) to specify `--redwood-like` before initializing your cluster.
- When invoking `initdb` to initialize your cluster compatible with Oracle mode, include the `--redwood-like` option or `--no-redwood-compat` option to initialize your cluster in Oracle non-compatible mode.

See Configuration parameters compatible with Oracle databases and Managing an EDB Postgres Advanced Server installation for more information about the installation options supported by the EDB Postgres Advanced Server installers.

2.2 Stored procedural language

EDB Postgres Advanced Server supports a highly productive procedural language that allows you to write custom procedures, functions, triggers and packages. The procedural language:

- complements the SQL language and built-in packages.
- provides a seamless development and testing environment.
- allows you to create reusable code.

See Database Compatibility: Stored Procedural Language for more information.

2.3 Optimizer hints

When you invoke a `DELETE` , `INSERT` , `SELECT` , or `UPDATE` command, the server generates a set of execution plans; after analyzing those execution plans, the server selects a plan that generally returns the result set in the least amount of time. The server's choice of plan is dependent upon several factors:

- The estimated execution cost of data handling operations.
- Parameter values assigned to parameters in the Query Tuning section of the `postgresql.conf` file.
- Column statistics that have been gathered by the `ANALYZE` command.

As a rule, the query planner selects the least expensive plan. You can use an optimizer hint to influence the server as it selects a query plan.

An optimizer hint is a directive (or multiple directives) embedded in a comment-like syntax that immediately follows a `DELETE` , `INSERT` , `SELECT` or `UPDATE` command. Keywords in the comment instruct the server to employ or avoid a specific plan when producing the result set. See *Optimizer hints* for more information.

2.4 Data dictionary views

EDB Postgres Advanced Server includes a set of views that provide information about database objects in a manner compatible with the Oracle data dictionary views. See *Database compatibility for Oracle developers: catalog views* for detailed information about the views available with EDB Postgres Advanced Server.

2.5 dblink_ora

`dblink_ora` provides an OCI-based database link that allows you to `SELECT` , `INSERT` , `UPDATE` or `DELETE` data stored on an Oracle system from within EDB Postgres Advanced Server. See *dblink_ora* for detailed information about using `dblink_ora` , and the supported functions and procedures.

2.6 Profile management

EDB Postgres Advanced Server supports compatible SQL syntax for profile management. Profile management commands allow a database superuser to create and manage named *profiles*. Each profile defines rules for password management that augment password and md5 authentication. The rules in a profile can:

- count failed login attempts
- lock an account due to excessive failed login attempts
- mark a password for expiration
- define a grace period after a password expiration
- define rules for password complexity
- define rules that limit password re-use

A profile is a named set of attributes that allow you to easily manage a group of roles that share comparable authentication requirements. If password requirements change, you can modify the profile to have the new requirements applied to each user that is associated with that profile.

After creating the profile, you can associate the profile with one or more users. When a user connects to the server, the server enforces the profile that is associated with their login role. Profiles are shared by all databases within a cluster,

but each cluster may have multiple profiles. A single user with access to multiple databases uses the same profile when connecting to each database within the cluster.

See *Profile management* for information about using profile management commands.

2.7 Built-in packages

EDB Postgres Advanced Server supports a number of built-in packages that provide compatibility with Oracle procedures and functions.

Package name	Description
DBMS_ALERT	The DBMS_ALERT package provides the capability to register for, send, and receive alerts.
DBMS_AQ	The DBMS_AQ package provides message queueing and processing for EDB Postgres Advanced Server.
DBMS_AQADM	The DBMS_AQADM package provides supporting procedures for Advanced Queueing functionality.
DBMS_CRYPTO	The DBMS_CRYPTO package provides functions and procedures that allow you to encrypt or decrypt RAW, BLOB or CLOB data. You can also use DBMS_CRYPTO functions to generate cryptographically strong random values.
DBMS_JOB	The DBMS_JOB package provides for the creation, scheduling, and managing of jobs.
DBMS_LOB	The DBMS_LOB package provides the capability to operate on large objects.
DBMS_LOCK	EDB Postgres Advanced Server provides support for the DBMS_LOCK.SLEEP procedure.
DBMS_MVIEW	Use procedures in the DBMS_MVIEW package to manage and refresh materialized views and their dependencies.
DBMS_OUTPUT	The DBMS_OUTPUT package provides the capability to send messages to a message buffer, or get messages from the message buffer.
DBMS_PIPE	The DBMS_PIPE package provides the capability to send messages through a pipe within or between sessions connected to the same database cluster.
DBMS_PROFILER	The DBMS_PROFILER package collects and stores performance information about the PL/pgSQL and SPL statements that are executed during a performance profiling session.
DBMS_RANDOM	The DBMS_RANDOM package provides a number of methods to generate random values.
DBMS_REDACT	The DBMS_REDACT package enables the redacting or masking of data that is returned by a query.
DBMS_RLS	The DBMS_RLS package enables the implementation of Virtual Private Database on certain EDB Postgres Advanced Server database objects.
DBMS_SCHEDULER	The DBMS_SCHEDULER package provides a way to create and manage jobs, programs and job schedules.
DBMS_SESSION	EDB Postgres Advanced Server provides support for the DBMS_SESSION.SET_ROLE procedure.
DBMS_SQL	The DBMS_SQL package provides an application interface to the EDB dynamic SQL functionality.
DBMS_UTILITY	The DBMS_UTILITY package provides various utility programs.
UTL_ENCODE	The UTL_ENCODE package provides a way to encode and decode data.
UTL_FILE	The UTL_FILE package provides the capability to read from, and write to files on the operating system's file system.
UTL_HTTP	The UTL_HTTP package provides a way to use the HTTP or HTTPS protocol to retrieve information found at an URL.

continues on next page

Table 1 – continued from previous page

Package name	Description
UTL_MAIL	The UTL_MAIL package provides the capability to manage e-mail.
UTL_RAW	The UTL_RAW package allows you to manipulate or retrieve the length of raw data types.
UTL_SMTP	The UTL_SMTP package provides the capability to send e-mails over the Simple Mail Transfer Protocol (SMTP).
UTL_URL	The UTL_URL package provides a way to escape illegal and reserved characters within an URL.

See *Built-in packages* for detailed information about the procedures and functions available within each package.

2.8 Open Client Library

The Open Client Library provides application interoperability with the Oracle Call Interface – an application that was formerly “locked in” can now work with either an EDB Postgres Advanced Server or an Oracle database with minimal to no changes to the application code. The EDB implementation of the Open Client Library is written in C.

The following diagram compares the Open Client Library and Oracle Call Interface application stacks.

For detailed information about the functions supported by the Open Client Library, see EDB OCL Connector .

2.9 Utilities

For detailed information about the compatible syntax supported by the utilities listed below, see Database Compatibility for Oracle Developers: Tools and Utilities .

2.9.1 EDB*Plus

EDB*Plus is a utility program that provides a command line user interface to the EDB Postgres Advanced Server that is familiar to Oracle developers and users. EDB*Plus accepts SQL commands, SPL anonymous blocks, and EDB*Plus commands.

EDB*Plus allows you to:

- Query certain database objects
- Execute stored procedures
- Format output from SQL commands
- Execute batch scripts
- Execute OS commands
- Record output

See EDBPlus for more information.

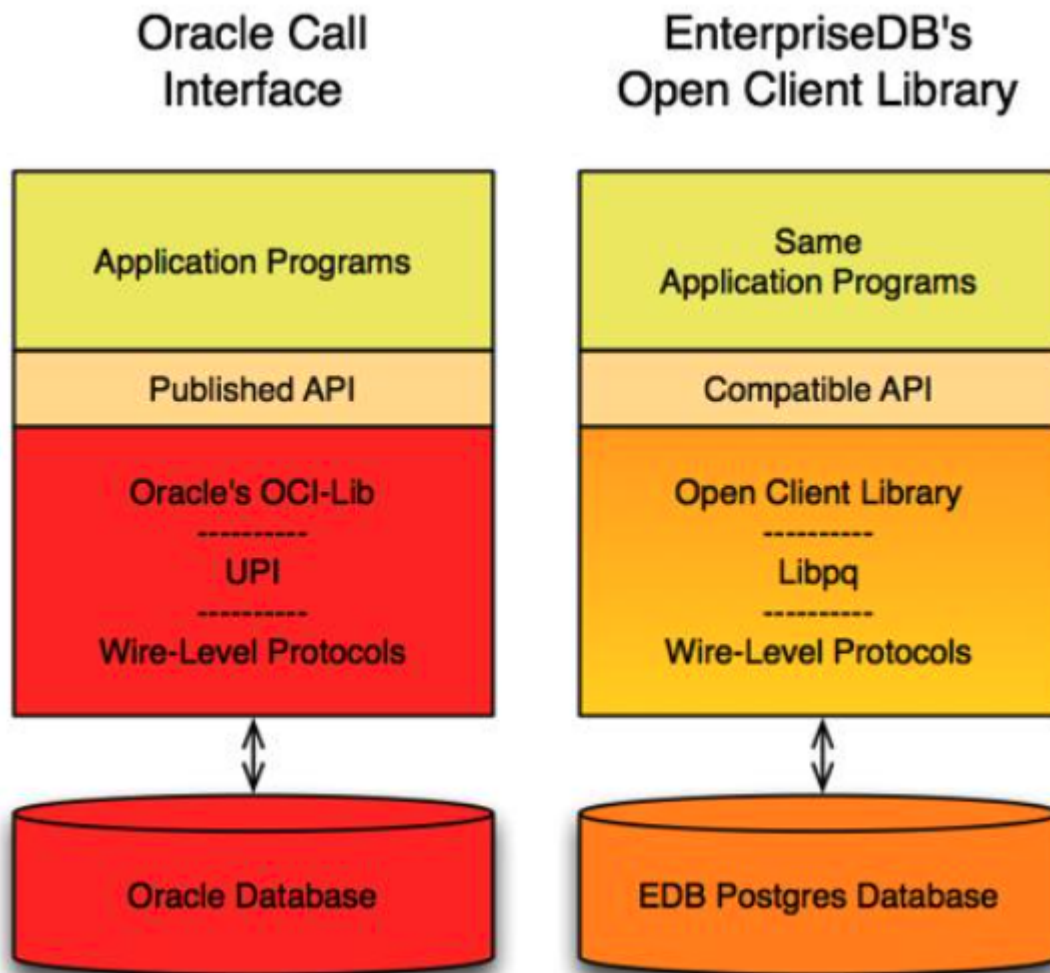


Fig. 1: The Open Client Library

2.9.2 EDB*Loader

EDB*Loader is a high-performance bulk data loader that provides an interface compatible with Oracle databases for EDB Postgres Advanced Server. The EDB*Loader command line utility loads data from an input source, typically a file, into one or more tables using a subset of the parameters offered by Oracle SQL*Loader.

EDB*Loader features include:

- Support for the Oracle SQL*Loader data loading methods - conventional path load, direct path load, and parallel direct path load
- Oracle SQL*Loader compatible syntax for control file directives
- Input data with delimiter-separated or fixed-width fields
- Bad file for collecting rejected records
- Loading of multiple target tables
- Discard file for collecting records that don't meet the selection criteria of any target table
- Data loading from standard input and remote loading

2.9.3 EDB*Wrap

The EDB*Wrap utility protects proprietary source code and programs (functions, stored procedures, triggers, and packages) from unauthorized scrutiny. The EDB*Wrap program translates a file that contains SPL or PL/pgSQL source code (the plaintext) into a file that contains the same code in a form that is nearly impossible to read. Once you have the obfuscated form of the code, you can send that code to EDB Postgres Advanced Server and it stores those programs in obfuscated form. While EDB*Wrap does obscure code, table definitions are still exposed.

Everything you wrap is stored in obfuscated form. If you wrap an entire package, the package body source, as well as the prototypes contained in the package header and the functions and procedures contained in the package body are stored in obfuscated form.

2.9.4 Dynamic Runtime Instrumentation Tools Architecture (DRITA)

The Dynamic Runtime Instrumentation Tools Architecture (DRITA) allows a DBA to query catalog views to determine the *wait events* that affect the performance of individual sessions or the system as a whole. DRITA records the number of times each event occurs as well as the time spent waiting; you can use this information to diagnose performance problems. DRITA offers this functionality, while consuming minimal system resources.

DRITA compares *snapshots* to evaluate the performance of a system. A snapshot is a saved set of system performance data at a given point in time. Each snapshot is identified by a unique ID number; you can use snapshot ID numbers with DRITA reporting functions to return system performance statistics.

2.10 ECPGPlus

EDB has enhanced ECPG (the PostgreSQL pre-compiler) to create ECPGPlus. ECPGPlus allows you to include embedded SQL commands in C applications; when you use ECPGPlus to compile an application that contains embedded SQL commands, the SQL code is syntax-checked and translated into C.

ECPGPlus supports Pro*C syntax in C programs when connected to an EDB Postgres Advanced Server database. ECPGPlus supports:

- Oracle Dynamic SQL – Method 4 (ODS-M4)

- Pro*C compatible anonymous blocks
- A CALL statement compatible with Oracle databases

See *ECPGPlus* for information.

2.11 Table partitioning

In a partitioned table, one logically large table is broken into smaller physical pieces. Partitioning can provide several benefits:

- Query performance can be improved dramatically in certain situations, particularly when most of the heavily accessed rows of the table are in a single partition or a small number of partitions. Partitioning allows you to omit the partition column from the front of an index, reducing index size and making it more likely that the heavily used parts of the index fits in memory.
- When a query or update accesses a large percentage of a single partition, performance may improve because the server performs a sequential scan of the partition instead of using an index and random access reads scattered across the whole table.
- A bulk load (or unload) can be implemented by adding or removing partitions, if you plan that requirement into the partitioning design. `ALTER TABLE` is far faster than a bulk operation. It also entirely avoids the `VACUUM` overhead caused by a bulk `DELETE`.
- Seldom-used data can be migrated to less-expensive (or slower) storage media.

Table partitioning is worthwhile only when a table would otherwise be very large. The exact point at which a table benefits from partitioning depends on the application; a good rule of thumb is that the size of the table should exceed the physical memory of the database server.

For information about database compatibility features supported by EDB Postgres Advanced Server, see Database Compatibility: Table Partitioning .

DATABASE ADMINISTRATION

This chapter describes the features that aid in the management and administration of EDB Postgres Advanced Server databases.

configuration_parameters index_advisor sql_profiler pgsnmpd edb_audit_logging unicode_collation_algorithm

3.1 Configuration parameters

The EDB Postgres Advanced Server configuration parameters control various aspects of the database server's behavior and environment such as data file and log file locations, connection, authentication and security settings, resource allocation and consumption, archiving and replication settings, error logging and statistics gathering, optimization and performance tuning, and locale and formatting settings

Configuration parameters that apply only to EDB Postgres Advanced Server are noted in the *Summary of configuration parameters* .

You can find more information about configuration parameters in the
[PostgreSQL core documentation](#) .

setting_new_parameters summary_of_configuration_parameters configuration_parameters_by_functionality

3.1.1 Setting configuration parameters

Set each configuration parameter using a name/value pair. Parameter names are not case sensitive. The parameter name is typically separated from its value by an optional equals sign (=).

This an example shows some configuration parameter settings in the `postgresql.conf` file:

```
# This is a comment

log_connections = yes
log_destination = syslog
search_path = "$user", public
shared_buffers = 128MB
```

Parameter values are specified as one of five types:

- Boolean. Acceptable values are `on` , `off` , `true` , `false` , `yes` , `no` , `1` , `0` , or any unambiguous prefix of these.
- Integer. Number without a fractional part.
- Floating point. Number with an optional fractional part separated by a decimal point.

- String. Text value enclosed in single quotes if the value isn't a simple identifier or number, that is, the value contains special characters such as spaces or other punctuation marks.
- Enum. Specific set of string values. The allowed values can be found in the system view `pg_settings.enumvals`. Enum values are not case sensitive.

Some settings specify a memory or time value. Each of these has an implicit unit, which is kilobytes, blocks (typically 8 kilobytes), milliseconds, seconds, or minutes. You can find default units by referencing the system view `pg_settings.unit`. You can specify a different explicitly.

Valid memory units are kB (kilobytes), MB (megabytes), and GB (gigabytes). Valid time units are ms (milliseconds), s (seconds), min (minutes), h (hours), and d (days). The multiplier for memory units is 1024.

You can set the configuration parameter settings in a number of different ways:

- A number of parameter settings are set when the EDB Postgres Advanced Server database product is built. These are read-only parameters, and you can't change their values. A couple of parameters are also permanently set for each database when the database is created. These parameters are read-only and you can't later change them for the database.
- The initial settings for almost all configurable parameters across the entire database cluster are listed in the configuration file `postgresql.conf`. These settings are put into effect upon database server start or restart. You can override some of these initial parameter settings. All configuration parameters have built-in default settings that are in effect if not explicitly overridden.
- Configuration parameters in the `postgresql.conf` file are overridden when the same parameters are included in the `postgresql.auto.conf` file. The `ALTER SYSTEM` command is used to manage the configuration parameters in the `postgresql.auto.conf` file.
- You can modify parameter settings in the configuration file while the database server is running. If the configuration file is then reloaded (meaning a `SIGHUP` signal is issued), for certain parameter types, the changed parameters settings immediately take effect. For some of these parameter types, the new settings are available in a currently running session immediately after the reload. For other of these parameter types, you must start a new session to use the new settings. And yet for other parameter types, modified settings don't take effect until the database server is stopped and restarted. See the [PostgreSQL core documentation](#) for information on how to reload the configuration file:
- You can use the SQL commands `ALTER DATABASE`, `ALTER ROLE`, or `ALTER ROLE IN DATABASE` to modify certain parameter settings. The modified parameter settings take effect for new sessions after you execute the command. `ALTER DATABASE` affects new sessions connecting to the specified database. `ALTER ROLE` affects new sessions started by the specified role. `ALTER ROLE IN DATABASE` affects new sessions started by the specified role connecting to the specified database. Parameter settings established by these SQL commands remain in effect indefinitely, across database server restarts, overriding settings established by the other methods. Parameter settings established using the `ALTER DATABASE`, `ALTER ROLE`, or `ALTER ROLE IN DATABASE` commands can be changed only by either:

- Reissuing these commands with a different parameter value.
- Issuing these commands using either of the `'SET parameter TO DEFAULT'` clause or the `'RESET parameter'` clause. These clauses change the parameter back to using the setting set by the other methods. See the `'PostgreSQL core documentation <https://www.postgresql.org/docs/current/sql-commands.html>'` for the exact syntax of these SQL commands.

- You can make changes for certain parameter settings for the duration of individual sessions using the `PGOPTIONS` environment variable or by using the `SET` command in the EDB-PSQL or PSQL command-line programs. Parameter settings made this way override settings established using any of the methods discussed earlier, but only during that session.

Modifying the postgresql.conf file

The configuration parameters in the `postgresql.conf` file specify server behavior with regard to auditing, authentication, encryption, and other behaviors. On Linux and Windows hosts, the `postgresql.conf` file resides in the data directory under your EDB Postgres Advanced Server installation.

Parameters that are preceded by a pound sign (#) are set to their default value. To change a parameter value, remove the pound sign and enter a new value. After setting or changing a parameter, you must either reload or restart the server for the new parameter value to take effect.

In the `postgresql.conf` file, some parameters contain comments that indicate `change requires restart`. To view a list of the parameters that require a server restart, execute the following query at the `psql` command line:

```
SELECT name FROM pg_settings WHERE context = postmaster;
```

Modifying the pg_hba.conf file

Appropriate authentication methods provide protection and security. Entries in the `pg_hba.conf` file specify the authentication methods that the server uses with connecting clients. Before connecting to the server, you might need to modify the authentication properties specified in the `pg_hba.conf` file.

When you invoke the `initdb` utility to create a cluster, the utility creates a `pg_hba.conf` file for that cluster that specifies the type of authentication required from connecting clients. You can modify this file. After modifying the authentication settings in the `pg_hba.conf` file, restart the server and apply the changes.

For more information about authentication and modifying the `pg_hba.conf` file, see the [PostgreSQL core documentation](#).

When the server receives a connection request, it verifies the credentials provided against the authentication settings in the `pg_hba.conf` file before allowing a connection to a database. To log the `pg_hba.conf` file entry to authenticate a connection to the server, set the `log_connections` parameter to `ON` in the `postgresql.conf` file.

A record specifies a connection type, database name, user name, client IP address, and the authentication method to authorize a connection upon matching these parameters in the `pg_hba.conf` file. Once the connection to a server is authorized, you can see the matched line number and the authentication record from the `pg_hba.conf` file.

This example shows a log detail for a valid `pg_hba.conf` entry after successful authentication:

```
2020-05-08 10:42:17 IST LOG:  connection received: host=[local]
2020-05-08 10:42:17 IST LOG:  connection authorized: user=u1 database=edb
application_name=psql
2020-05-08 10:42:17 IST DETAIL:  Connection matched pg_hba.conf line 84:
"local    all                all                md5"
```

3.1.2 Summary of configuration parameters

This summary table lists all EDB Postgres Advanced Server configuration parameters along with a number of key attributes of the parameters.

These attributes are described by the following columns of the summary table:

Parameter. Configuration parameter name.

Scope of effect. Scope of effect of the configuration parameter setting:

- **Cluster**. Setting affects the entire database cluster, that is, all databases managed by the database server instance.

- **Database** . Setting can vary by database and is established when the database is created. Applies to a small number of parameters related to locale settings.
- **Session** . Setting can vary down to the granularity of individual sessions.

In other words, different settings can be made for the following entities in which the latter settings in this list override prior ones:

- The entire database cluster
- Specific databases in the database cluster
- Specific roles
- Specific roles when connected to specific databases
- A specific session

When takes effect. When a changed parameter setting takes effect.

- **Preset** . Established when the EDB Postgres Advanced Server product is built or a particular database is created. This is a read-only parameter and can't be changed.
- **Restart** . You must restart the database server.
- **Reload** . You must reload configuration file or restart the database server.
- **Immediate** . Immediately effective in a session if the `PGOPTIONS` environment variable or the `SET` command is used to change the setting in the current session.

Effective in new sessions if `ALTER DATABASE` , `ALTER ROLE` , or `ALTER ROLE IN DATABASE` commands are used to change the setting.

Authorized user. Type of operating system account or database role that must be used to put the parameter setting into effect.

- **EPAS service account** . EDB Postgres Advanced Server service account (`enterprisedb` for an installation compatible with Oracle databases, `postgres` for a PostgreSQL compatible mode installation).
- **Superuser** . Database role with superuser privileges.
- **User** . Any database role with permissions on the affected database object, that is, the database or role to alter with the `ALTER` command.
- **n/a** – No user can change the parameter setting.

Description. Brief description of the configuration parameter.

EPAS only. Configuration parameter applies to EDB Postgres Advanced Server only. No entry in this column indicates the configuration parameter applies to PostgreSQL as well.

Note: A number of parameters can never be altered. These are designated as **Note: For internal use only** in the Description column.

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>allow_system_table_mods</i>	Cluster	Restart	EPAS service account	Allows modifications of the structure of system tables.	
<i>application_name</i>	Session	Immediate	User	Sets the application name to be reported in statistics and logs.	
<i>archive_command</i>	Cluster	Reload	EPAS service account	Sets the shell command called to archive a WAL file.	
<i>archive_mode</i>	Cluster	Restart	EPAS service account	Allows archiving of WAL files using <i>archive_command</i> .	
<i>archive_timeout</i>	Cluster	Reload	EPAS service account	Forces a switch to the next xlog file if a new file has not been started within N seconds.	
<i>array_nulls</i>	Session	Immediate	User	Enables input of NULL elements in arrays.	
<i>authentication_timeout</i>	Cluster	Reload	EPAS service account	Sets the maximum allowed time to complete client authentication.	
<i>autovacuum</i>	Cluster	Reload	EPAS service account	Starts the autovacuum subprocess.	
<i>autovacuum_analyze_scale_factor</i>	Cluster	Reload	EPAS service account	Number of tuple inserts, updates, or deletes prior to analyze as a fraction of <i>reltuples</i> .	
<i>autovacuum_analyze_threshold</i>	Cluster	Reload	EPAS service account	Minimum number of tuple inserts, updates, or deletes prior to analyze.	
<i>autovacuum_freeze_max_age</i>	Cluster	Restart	EPAS service account	Age at which to autovacuum a table to prevent transaction ID wraparound.	
<i>autovacuum_max_workers</i>	Cluster	Restart	EPAS service account	Sets the maximum number of simultaneously running autovacuum worker processes.	
<i>autovacuum_multixact_freeze_max_age</i>	Cluster	Restart	EPAS service account	Multixact age at which to autovacuum a table to prevent multixact wraparound.	
<i>autovacuum_naptime</i>	Cluster	Reload	EPAS service account	Time to sleep between autovacuum runs.	
<i>autovacuum_vacuum_cost_delay</i>	Cluster	Reload	EPAS service account	Vacuum cost delay in milliseconds, for autovacuum.	

continues on next page

Table 1 – continued from previous page

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>autovacuum_vacuum_cost_limit</i>	Cluster	Reload	EPAS service count	Vacuum cost amount available before napping, for autovacuum.	
<i>autovacuum_vacuum_scale_factor</i>	Cluster	Reload	EPAS service count	Number of tuple updates or deletes prior to vacuum as a fraction of <i>rel-tuples</i> .	
<i>autovacuum_vacuum_threshold</i>	Cluster	Reload	EPAS service count	Minimum number of tuple updates or deletes prior to vacuum.	
<i>autovacuum_work_mem</i>	Cluster	Reload	EPAS service count	Sets the maximum memory to be used by each autovacuum worker process.	
<i>backslash_quote</i>	Session	Immediate	User	Sets whether “” is allowed in string literals.	
<i>bgwriter_delay</i>	Cluster	Reload	EPAS service count	Background writer sleep time between rounds.	
<i>bgwriter_lru_maxpages</i>	Cluster	Reload	EPAS service count	Background writer maximum number of LRU pages to flush per round.	
<i>bgwriter_lru_multiplier</i>	Cluster	Reload	EPAS service count	Multiple of the average buffer usage to free per round.	
<i>block_size</i>	Cluster	Preset	n/a	Shows the size of a disk block.	
<i>bonjour</i>	Cluster	Restart	EPAS service count	Enables advertising the server via Bonjour.	
<i>bonjour_name</i>	Cluster	Restart	EPAS service count	Sets the Bonjour service name.	
<i>bytea_output</i>	Session	Immediate	User	Sets the output format for <i>bytea</i> .	
<i>check_function_bodies</i>	Session	Immediate	User	Checks function bodies during <i>CREATE FUNCTION</i> .	
<i>checkpoint_completion_target</i>	Cluster	Reload	EPAS service count	Time spent flushing dirty buffers during checkpoint, as fraction of checkpoint interval.	
<i>checkpoint_segments</i>	Deprecated	Deprecated	Deprecated	Specifies a value for the parameter prevents the server from starting.	
<i>checkpoint_timeout</i>	Cluster	Reload	EPAS service count	Sets the maximum time between automatic WAL checkpoints.	

continues on next page

Table 1 – continued from previous page

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>checkpoint_warning</i>	Cluster	Reload	EPAS service account	Enables warnings if checkpoint segments are filled more frequently than this.	
<i>client_encoding</i>	Session	Immediate	User	Sets the client's character set encoding.	
<i>client_min_messages</i>	Session	Immediate	User	Sets the message levels that are sent to the client.	
<i>commit_delay</i>	Session	Immediate	Superuser	Sets the delay in microseconds between transaction commit and flushing WAL to disk.	
<i>commit_siblings</i>	Session	Immediate	User	Sets the minimum concurrent open transactions before performing <i>commit_delay</i> .	
<i>config_file</i>	Cluster	Restart	EPAS service account	Sets the server's main configuration file.	
<i>constraint_exclusion</i>	Session	Immediate	User	Enables the planner to use constraints to optimize queries.	
<i>cpu_index_tuple_cost</i>	Session	Immediate	User	Sets the planner's estimate of the cost of processing each index entry during an index scan.	
<i>cpu_operator_cost</i>	Session	Immediate	User	Sets the planner's estimate of the cost of processing each operator or function call.	
<i>cpu_tuple_cost</i>	Session	Immediate	User	Sets the planner's estimate of the cost of processing each tuple (row).	
<i>cursor_tuple_fraction</i>	Session	Immediate	User	Sets the planner's estimate of the fraction of a cursor's rows to retrieve.	
<i>custom_variable_classes</i>	Cluster	Reload	EPAS service account	Deprecated in EDB Postgres Advanced Server 9.2.	X
<i>data_checksums</i>	Cluster	Preset	n/a	Shows whether data checksums are turned on for this cluster.	
<i>data_directory</i>	Cluster	Restart	EPAS service account	Sets the server's data directory.	
<i>datestyle</i>	Session	Immediate	User	Sets the display format for date and time values.	

continues on next page

Table 1 – continued from previous page

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>db_dialect</i>	Session	Immediate	User	Sets the precedence of built-in namespaces.	X
<i>dbms_alert.max_alerts</i>	Cluster	Restart	EPAS service account	Sets maximum number of alerts.	X
<i>dbms_pipe.total_message_buffer</i>	Cluster	Restart	EPAS service account	Specifies the total size of the buffer used for the <i>DBMS_PIPE</i> package.	X
<i>db_user_namespace</i>	Cluster	Reload	EPAS service account	Enables per-database user names.	
<i>deadlock_timeout</i>	Session	Immediate	Superuser	Sets the time to wait on a lock before checking for deadlock.	
<i>debug_assertions</i>	Cluster	Preset	n/a	Turns on various assertion checks. (Not supported in EPAS builds.)	
<i>debug_pretty_print</i>	Session	Immediate	User	Indents parse and plan tree displays.	
<i>debug_print_parse</i>	Session	Immediate	User	Logs each query's parse tree.	
<i>debug_print_plan</i>	Session	Immediate	User	Logs each query's execution plan.	
<i>debug_print_rewritten</i>	Session	Immediate	User	Logs each query's rewritten parse tree.	
<i>default_heap_fillfactor</i>	Session	Immediate	User	Creates new tables with this heap fill factor by default.	X
<i>default_statistics_target</i>	Session	Immediate	User	Sets the default statistics target.	
<i>default_tablespace</i>	Session	Immediate	User	Sets the default tablespace to create tables and indexes in.	
<i>default_text_search_config</i>	Session	Immediate	User	Sets default text search configuration.	
<i>default_transaction_deferrable</i>	Session	Immediate	User	Sets the default deferrable status of new transactions.	
<i>default_transaction_isolation</i>	Session	Immediate	User	Sets the transaction isolation level of each new transaction.	
<i>default_transaction_read_only</i>	Session	Immediate	User	Sets the default read-only status of new transactions.	
<i>default_with_oids</i>	Session	Immediate	User	Creates new tables with OIDs by default.	
<i>default_with_rowids</i>	Session	Immediate	User	Creates new tables with ROWID support (ROWIDs with indexes) by default.	X

continues on next page

Table 1 – continued from previous page

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>dynamic_library_path</i>	Session	Immediate	Superuser	Sets the path for dynamically loadable modules.	
<i>dynamic_shared_memory_type</i>	Cluster	Restart	EPAS service account	Selects the dynamic shared memory implementation used.	
<i>edb_audit</i>	Cluster	Reload	EPAS service account	Enables EDB Auditing to create audit reports in XML or CSV format.	X
<i>edb_audit_archiver</i>	Cluster	Restart	EPAS service account	Enables audit log archiver process.	X
<i>edb_audit_archiver_timeout</i>	Cluster	Reload	EPAS service account	Checks the audit log files based on the <i>edb_audit_archiver_timeout</i> parameter.	X
<i>edb_audit_archiver_filename_prefix</i>	Cluster	Reload	EPAS service account	Determines audit log files with <i>edb_audit_archiver_filename_prefix</i> in <i>edb_audit_directory</i> are eligible for compression and/or expiration; the parameter must align with <i>edb_audit_filename</i> .	X
<i>edb_audit_archiver_compression</i>	Cluster	Reload	EPAS service account	Shows time in seconds after which the audit logs are eligible for compression.	X
<i>edb_audit_archiver_compression_limit</i>	Cluster	Reload	EPAS service account	Shows total size in megabytes after which the audit logs are eligible for compression.	X
<i>edb_audit_archiver_compression_command</i>	Cluster	Reload	EPAS service account	Compresses the audit log files.	X
<i>edb_audit_archiver_compression_suffix</i>	Cluster	Reload	EPAS service account	Determines suffix for an already compressed log file. The parameter must align with <i>edb_audit_archiver_compression_command</i> .	X
<i>edb_audit_archiver_expire_timeout</i>	Cluster	Reload	EPAS service account	Shows time in seconds after which the audit logs are eligible for expiration.	X
<i>edb_audit_archiver_expire_size_limit</i>	Cluster	Reload	EPAS service account	Shows total size in megabytes after which the audit logs are eligible for expiration.	X

continues on next page

Table 1 – continued from previous page

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>edb_audit_archiver_expire</i>	Cluster	Reload	EPAS service count	Execute on an expired audit log before removing it.	X
<i>edb_audit_archiver_sort_file</i>	Cluster	Reload	EPAS service count	Identifies the oldest log file and sorts them alphabetically or based on <i>mtime</i> .	X
<i>edb_audit_connect</i>	Cluster	Reload	EPAS service count	Audits each successful connection.	X
<i>edb_audit_destination</i>	Cluster	Reload	EPAS service count	Sets <i>edb_audit_directory</i> or <i>syslog</i> as the destination directory for audit files. The <i>syslog</i> setting is only valid for a Linux system.	X
<i>edb_audit_directory</i>	Cluster	Reload	EPAS service count	Sets the destination directory for audit files.	X
<i>edb_audit_disconnect</i>	Cluster	Reload	EPAS service count	Audits end of a session.	X
<i>edb_audit_filename</i>	Cluster	Reload	EPAS service count	Sets the file name pattern for audit files.	X
<i>edb_audit_rotation_day</i>	Cluster	Reload	EPAS service count	Automatic rotation of log files based on day of week.	X
<i>edb_audit_rotation_seconds</i>	Cluster	Reload	EPAS service count	Automatic log file rotation occurs after N seconds.	X
<i>edb_audit_rotation_size</i>	Cluster	Reload	EPAS service count	Automatic log file rotation occurs after N Megabytes.	X
<i>edb_audit_statement</i>	Cluster	Reload	EPAS service count	Sets the type of statements to audit.	X
<i>edb_audit_tag</i>	Session	Immediate	User	Specifies a tag to include in the audit log.	X
<i>edb_connectby_order</i>	Session	Immediate	User	Sorts results of <i>CONNECT BY</i> queries with no <i>ORDER BY</i> to depth-first order. Note: For internal use only.	X
<i>edb_custom_plan_tries</i>	Session	Immediate	User	Specifies the number of custom execution plans considered by the planner before the planner selects a generic execution plan.	X

continues on next page

Table 1 – continued from previous page

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>edb_data_redaction</i>	Session	Immediate	User	Enables data redaction.	X
<i>edb_dynatune</i>	Cluster	Restart	EPAS service account	Sets the edb utilization percentage.	X
<i>edb_dynatune_profile</i>	Cluster	Restart	EPAS service account	Sets the workload profile for dynatune.	X
<i>edb_enable_pruning</i>	Session	Immediate	User	Enables the planner to early-prune partitioned tables.	X
<i>edb_log_every_bulk_value</i>	Session	Immediate	Superuser	Sets the statements logged for bulk processing.	X
<i>edb_max_resource_groups</i>	Cluster	Restart	EPAS service account	Specifies the maximum number of resource groups for simultaneous use.	X
<i>edb_max_spins_per_delay</i>	Cluster	Restart	EPAS service account	Specifies the number of times a session spins while waiting for a lock.	X
<i>edb_redwood_date</i>	Session	Immediate	User	Determines whether <i>DATE</i> should behave like a <i>TIMESTAMP</i> or not.	X
<i>edb_redwood_greatest_least</i>	Session	Immediate	User	Determines how <i>GREATEST</i> and <i>LEAST</i> functions should handle NULL parameters.	X
<i>edb_redwood_raw_names</i>	Session	Immediate	User	Returns the unmodified name stored in the PostgreSQL system catalogs from Redwood interfaces.	X
<i>edb_redwood_strings</i>	Session	Immediate	User	Treats <i>NULL</i> as an empty string when concatenated with a text value.	X
<i>edb_resource_group</i>	Session	Immediate	User	Specifies the resource group to be used by the current process.	X
<i>edb_sql_protect.enabled</i>	Cluster	Reload	EPAS service account	Defines whether SQL/Protect should track queries or not.	X
<i>edb_sql_protect.level</i>	Cluster	Reload	EPAS service account	Defines the behavior of SQL/Protect when an event is found.	X
<i>edb_sql_protect.max_protected_queries</i>	Cluster	Restart	EPAS service account	Sets the maximum number of relations protected by SQL/Protect per role.	X
<i>edb_sql_protect.max_protected_queries</i>	Cluster	Restart	EPAS service account	Sets the maximum number of roles protected by SQL/Protect.	X

continues on next page

Table 1 – continued from previous page

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>edb_sql_protect_max_queries_per_cluster</i>	Cluster	Restart	EPAS service account	Sets the maximum number of offending queries to save by SQL/Protect.	X
<i>edb_stmt_level_tx</i>	Session	Immediate	User	Allows continuing on errors instead of requiring a transaction abort.	X
<i>edb_wait_states.directory</i>	Cluster	Restart	EPAS service account	Stores the EDB wait states logs in this directory.	X
<i>edb_wait_states.retention_period</i>	Cluster	Reload	EPAS service account	Deletes EDB wait state log files after retention period.	X
<i>edb_wait_states.sampling_interval</i>	Cluster	Reload	EPAS service account	The interval between two EDB wait state sampling cycles.	X
<i>edbldr.empty_csv_field</i>	Session	Immediate	Superuser	Specifies how EDB*Loader handles empty strings.	X
<i>effective_cache_size</i>	Session	Immediate	User	Sets the planner's assumption about the size of the disk cache.	
<i>effective_io_concurrency</i>	Session	Immediate	User	Number of simultaneous requests that can be handled efficiently by the disk subsystem.	
<i>enable_bitmapscan</i>	Session	Immediate	User	Enables the planner's use of bitmap-scan plans.	
<i>enable_hashagg</i>	Session	Immediate	User	Enables the planner's use of hashed aggregation plans.	
<i>enable_hashjoin</i>	Session	Immediate	User	Enables the planner's use of hash join plans.	
<i>enable_hints</i>	Session	Immediate	User	Enables optimizer hints in SQL statements.	X
<i>enable_indexonlyscan</i>	Session	Immediate	User	Enables the planner's use of index-only-scan plans.	
<i>enable_indexscan</i>	Session	Immediate	User	Enables the planner's use of index-scan plans.	
<i>enable_material</i>	Session	Immediate	User	Enables the planner's use of materialization.	
<i>enable_mergejoin</i>	Session	Immediate	User	Enables the planner's use of merge join plans.	

continues on next page

Table 1 – continued from previous page

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>enable_nestloop</i>	Session	Immediate	User	Enables the planner's use of nested-loop join plans.	
<i>enable_seqscan</i>	Session	Immediate	User	Enables the planner's use of sequential-scan plans.	
<i>enable_sort</i>	Session	Immediate	User	Enables the planner's use of explicit sort steps.	
<i>enable_tidscan</i>	Session	Immediate	User	Enables the planner's use of TID scan plans.	
<i>escape_string_warning</i>	Session	Immediate	User	Warns about backlash escapes in ordinary string literals.	
<i>event_source</i>	Cluster	Restart	EPAS service account	Sets the application name used to identify PostgreSQL messages in the event log.	
<i>exit_on_error</i>	Session	Immediate	User	Terminates session on any error.	
<i>external_pid_file</i>	Cluster	Restart	EPAS service account	Writes the postmaster PID to the specified file.	
<i>extra_float_digits</i>	Session	Immediate	User	Sets the number of digits displayed for floating-point values.	
<i>from_collapse_limit</i>	Session	Immediate	User	Sets the <i>FROM</i> -list size beyond which subqueries are not collapsed.	
<i>fsync</i>	Cluster	Reload	EPAS service account	Forces synchronization of updates to disk.	
<i>full_page_writes</i>	Cluster	Reload	EPAS service account	Writes full pages to WAL when first modified after a checkpoint.	
<i>geqo</i>	Session	Immediate	User	Enables genetic query optimization.	
<i>geqo_effort</i>	Session	Immediate	User	GEQO: effort is used to set the default for other GEQO parameters.	
<i>geqo_generations</i>	Session	Immediate	User	GEQO: number of iterations of the algorithm.	
<i>geqo_pool_size</i>	Session	Immediate	User	GEQO: number of individuals in the population.	
<i>geqo_seed</i>	Session	Immediate	User	GEQO: seed for random path selection.	
<i>geqo_selection_bias</i>	Session	Immediate	User	GEQO: selective pressure within the population.	

continues on next page

Table 1 – continued from previous page

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>geqo_threshold</i>	Session	Immediate	User	Sets the threshold of <i>FROM</i> items beyond which GEQO is used.	
<i>gin_fuzzy_search_limit</i>	Session	Immediate	User	Sets the maximum allowed result for exact search by GIN.	
<i>hba_file</i>	Cluster	Restart	EPAS service account	Sets the server's "hba" configuration file.	
<i>hot_standby</i>	Cluster	Restart	EPAS service account	Allows connections and queries during recovery.	
<i>hot_standby_feedback</i>	Cluster	Reload	EPAS service account	Allows feedback from a hot standby to the primary that avoids query conflicts.	
<i>huge_pages</i>	Cluster	Restart	EPAS service account	Uses huge pages on Linux.	
<i>icu_short_form</i>	Database	Preset	n/a	Shows the ICU collation order configuration.	X
<i>ident_file</i>	Cluster	Restart	EPAS service account	Sets the server's "ident" configuration file.	
<i>ignore_checksum_failure</i>	Session	Immediate	Superuser	Continues processing after a checksum failure.	
<i>ignore_system_indexes</i>	Cluster/Session	Reload/Immediate	EPAS service account/User	Disables reading from system indexes. (Can also be set with <i>PGOPTIONS</i> at session start.)	
<i>index_advisor.enabled</i>	Session	Immediate	User	Enables Index Advisor plugin.	X
<i>integer_datetimes</i>	Cluster	Preset	n/a	Datetimes are integer based.	
<i>intervalStyle</i>	Session	Immediate	User	Sets the display format for interval values.	
<i>join_collapse_limit</i>	Session	Immediate	User	Sets the <i>FROM</i> -list size beyond which <i>JOIN</i> constructs are not flattened.	
<i>krb_caseins_users</i>	Cluster	Reload	EPAS service account	Sets whether Kerberos and GSSAPI user names should be treated as case-insensitive.	
<i>krb_server_keyfile</i>	Cluster	Reload	EPAS service account	Sets the location of the Kerberos server key file.	
<i>lc_collate</i>	Database	Preset	n/a	Shows the collation order locale.	

continues on next page

Table 1 – continued from previous page

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>lc_ctype</i>	Database	Preset	n/a	Shows the character classification and case conversion locale.	
<i>lc_messages</i>	Session	Immediate	Superuser	Sets the language in which messages are displayed.	
<i>lc_monetary</i>	Session	Immediate	User	Sets the locale for formatting monetary amounts.	
<i>lc_numeric</i>	Session	Immediate	User	Sets the locale for formatting numbers.	
<i>lc_time</i>	Session	Immediate	User	Sets the locale for formatting date and time values.	
<i>listen_addresses</i>	Cluster	Restart	EPAS service account	Sets the host name or IP address(es) to listen to.	
<i>local_preload_libraries</i>	Cluster/Session	Reload/Immediate	EPAS service account/User	Lists shared libraries to preload into each backend. (Can also be set with <i>PGOPTIONS</i> at session start.)	
<i>lock_timeout</i>	Session	Immediate	User	Sets the maximum time allowed that a statement may wait for a lock.	
<i>lo_compat_privileges</i>	Session	Immediate	Superuser	Enables backward compatibility mode for privilege checks on large objects.	
<i>log_autovacuum_min_duration@cluster</i>		Reload	EPAS service account	Sets the minimum execution time above which autovacuum actions are logged.	
<i>log_checkpoints</i>	Cluster	Reload	EPAS service account	Logs each checkpoint.	
<i>log_connections</i>	Cluster/Session	Reload/Immediate	EPAS service account/User	Logs each successful connection. (Can also be set with <i>PGOPTIONS</i> at session start.)	
<i>log_destination</i>	Cluster	Reload	EPAS service account	Sets the destination for server log output.	
<i>log_directory</i>	Cluster	Reload	EPAS service account	Sets the destination directory for log files.	
<i>log_disconnections</i>	Cluster/Session	Reload/Immediate	EPAS service account/User	Logs end of a session, including duration. (Can also be set with <i>PGOPTIONS</i> at session start.)	

continues on next page

Table 1 – continued from previous page

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>log_duration</i>	Session	Immediate	Superuser	Logs the duration of each completed SQL statement.	
<i>log_error_verbosity</i>	Session	Immediate	Superuser	Sets the verbosity of logged messages.	
<i>log_executor_stats</i>	Session	Immediate	Superuser	Writes executor performance statistics to the server log.	
<i>log_file_mode</i>	Cluster	Reload	EPAS service account	Sets the file permissions for log files.	
<i>log_filename</i>	Cluster	Reload	EPAS service account	Sets the file name pattern for log files.	
<i>log_hostname</i>	Cluster	Reload	EPAS service account	Logs the host name in the connection logs.	
<i>log_line_prefix</i>	Cluster	Reload	EPAS service account	Controls information prefixed to each log line.	
<i>log_lock_waits</i>	Session	Immediate	Superuser	Logs long lock waits.	
<i>log_min_duration_statement</i>	Session	Immediate	Superuser	Sets the minimum execution time above which statements are logged.	
<i>log_min_error_statement</i>	Session	Immediate	Superuser	Causes all statements generating error at or above this level to be logged.	
<i>log_min_messages</i>	Session	Immediate	Superuser	Sets the message levels that are logged.	
<i>log_parser_stats</i>	Session	Immediate	Superuser	Writes parser performance statistics to the server log.	
<i>log_planner_stats</i>	Session	Immediate	Superuser	Writes planner performance statistics to the server log.	
<i>log_rotation_age</i>	Cluster	Reload	EPAS service account	Automatic log file rotation occurs after N minutes.	
<i>log_rotation_size</i>	Cluster	Reload	EPAS service account	Automatic log file rotation occurs after N kilobytes.	
<i>log_statement</i>	Session	Immediate	Superuser	Sets the type of statements logged.	
<i>log_statement_stats</i>	Session	Immediate	Superuser	Writes cumulative performance statistics to the server log.	
<i>log_temp_files</i>	Session	Immediate	Superuser	Logs the use of temporary files larger than this number of kilobytes.	

continues on next page

Table 1 – continued from previous page

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>log_timezone</i>	Cluster	Reload	EPAS service account	Sets the time zone to use in log messages.	
<i>log_truncate_on_rotation</i>	Cluster	Reload	EPAS service account	Truncates existing log files of same name during log rotation.	
<i>logging_collector</i>	Cluster	Restart	EPAS service account	Starts a subprocess to capture stderr output and/or csv logs into log files.	
<i>maintenance_work_mem</i>	Session	Immediate	User	Sets the maximum memory to be used for maintenance operations.	
<i>max_connections</i>	Cluster	Restart	EPAS service account	Sets the maximum number of current connections.	
<i>max_files_per_process</i>	Cluster	Restart	EPAS service account	Sets the maximum number of simultaneously open files for each server process.	
<i>max_function_args</i>	Cluster	Preset	n/a	Shows the maximum number of function arguments.	
<i>max_identifier_length</i>	Cluster	Preset	n/a	Shows the maximum identifier length.	
<i>max_index_keys</i>	Cluster	Preset	n/a	Shows the maximum number of index keys.	
<i>max_locks_per_transaction</i>	Cluster	Restart	EPAS service account	Sets the maximum number of locks per transaction.	
<i>max_pred_locks_per_transaction</i>	Cluster	Restart	EPAS service account	Sets the maximum number of predicate locks per transaction.	
<i>max_prepared_transactions</i>	Cluster	Restart	EPAS service account	Sets the maximum number of simultaneously prepared transactions.	
<i>max_replication_slots</i>	Cluster	Restart	EPAS service account	Sets the maximum number of simultaneously defined replication slots.	
<i>max_stack_depth</i>	Session	Immediate	Superuser	Sets the maximum stack depth, in kilobytes.	
<i>max_standby_archive_delay</i>	Cluster	Reload	EPAS service account	Sets the maximum delay before canceling queries when a hot standby server is processing archived WAL data.	

continues on next page

Table 1 – continued from previous page

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>max_standby_streaming_delay</i>	Cluster	Reload	EPAS service count	Sets the maximum delay before canceling queries when a hot standby server is processing streamed WAL data.	
<i>max_wal_senders</i>	Cluster	Restart	EPAS service count	Sets the maximum number of simultaneously running WAL sender processes.	
<i>max_wal_size</i>	Cluster	Reload	EPAS service count	Sets the maximum size to which the WAL grows between automatic WAL checkpoints. The default is 1GB.	
<i>max_worker_processes</i>	Cluster	Restart	EPAS service count	Maximum number of concurrent worker processes.	
<i>min_wal_size</i>	Cluster	Reload	EPAS service count	Sets the threshold at which WAL logs is recycled rather than removed. The default is 80 MB.	
<i>nlslength_semantics</i>	Session	Immediate	Superuser	Sets the semantics to use for char, varchar, varchar2 and long columns.	X
<i>odbc_lib_path</i>	Cluster	Restart	EPAS service count	Sets the path for ODBC library.	X
<i>optimizer_mode</i>	Session	Immediate	User	Default optimizer mode.	X
<i>oracle_home</i>	Cluster	Restart	EPAS service count	Sets the path for the Oracle home directory.	X
<i>password_encryption</i>	Session	Immediate	User	Encrypts passwords.	
<i>pg_prewarm.autoprewarm</i>	Cluster	Restart	EPAS service count	Enables the <i>autoprewarm</i> background worker.	X
<i>pg_prewarm.autoprewarm_interval</i>	Cluster	Reload	EPAS service count	Sets the minimum number of seconds after which <i>autoprewarm</i> dumps shared buffers.	X
<i>port</i>	Cluster	Restart	EPAS service count	Sets the TCP port on which the server listens.	
<i>post_auth_delay</i>	Cluster/Session	Reload/Immediate	EPAS service count/User	Waits N seconds on connection startup after authentication. (Can also be set with <i>PGOPTIONS</i> at session start.)	

continues on next page

Table 1 – continued from previous page

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>pre_auth_delay</i>	Cluster	Reload	EPAS service account	Waits N seconds on connection startup before authentication.	
<i>qreplace_function</i>	Session	Immediate	Superuser	The function to be used by Query Replace feature. Note: For internal use only.	X
<i>query_rewrite_enabled</i>	Session	Immediate	User	Skills dchild table scans if their constraints guarantee that no rows match the query.	X
<i>query_rewrite_integrity</i>	Session	Immediate	Superuser	Sets the degree to which query rewriting must be enforced.	X
<i>quote_all_identifiers</i>	Session	Immediate	User	When generating SQL fragments, quotes all identifiers.	
<i>random_page_cost</i>	Session	Immediate	User	Sets the planner's estimate of the cost of a nonsequentially fetched disk page.	
<i>restart_after_crash</i>	Cluster	Reload	EPAS service account	Reinitializes server after backend crash.	
<i>search_path</i>	Session	Immediate	User	Sets the schema search order for names that are not schema-qualified.	
<i>segment_size</i>	Cluster	Preset	n/a	Shows the number of pages per disk file.	
<i>seq_page_cost</i>	Session	Immediate	User	Sets the planner's estimate of the cost of a sequentially fetched disk page.	
<i>server_encoding</i>	Database	Preset	n/a	Sets the server (database) character set encoding.	
<i>server_version</i>	Cluster	Preset	n/a	Shows the server version.	
<i>server_version_num</i>	Cluster	Preset	n/a	Shows the server version as an integer.	
<i>session_preload_libraries</i>	Session	Immediate but only at connection start	Superuser	Lists shared libraries to preload into each backend.	
<i>session_replication_role</i>	Session	Immediate	Superuser	Sets the session's behavior for triggers and rewrite rules.	
<i>shared_buffers</i>	Cluster	Restart	EPAS service account	Sets the number of shared memory buffers used by the server.	

continues on next page

Table 1 – continued from previous page

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>shared_preload_libraries</i>	Cluster	Restart	EPAS service count	Lists shared libraries to preload into server.	
<i>sql_inheritance</i>	Session	Immediate	User	Causes subtables to be included by default in various commands.	
<i>ssl</i>	Cluster	Reload	EPAS service count	Enables SSL connections.	
<i>ssl_ca_file</i>	Cluster	Reload	EPAS service count	Location of the SSL certificate authority file.	
<i>ssl_cert_file</i>	Cluster	Reload	EPAS service count	Location of the SSL server certificate file.	
<i>ssl_ciphers</i>	Cluster	Reload	EPAS service count	Sets the list of allowed SSL ciphers.	
<i>ssl_crl_file</i>	Cluster	Reload	EPAS service count	Location of the SSL certificate revocation list file.	
<i>ssl_ecdh_curve</i>	Cluster	Reload	EPAS service count	Sets the curve to use for ECDH.	
<i>ssl_key_file</i>	Cluster	Reload	EPAS service count	Location of the SSL server private key file.	
<i>ssl_prefer_server_ciphers</i>	Cluster	Reload	EPAS service count	Gives priority to server ciphersuite order.	
<i>ssl_renegotiation_limit</i>	Session	Immediate	User	Sets the amount of traffic to send and receive before renegotiating the encryption keys.	
<i>standard_conforming_strings</i>	Session	Immediate	User	Causes '...' strings to treat backslashes literally.	
<i>statement_timeout</i>	Session	Immediate	User	Sets the maximum allowed duration of any statement.	
<i>stats_temp_directory</i>	Cluster	Reload	EPAS service count	Writes temporary statistics files to the specified directory.	
<i>superuser_reserved_connections</i>	Cluster	Restart	EPAS service count	Sets the number of connection slots reserved for superusers.	
<i>synchronize_seqscans</i>	Session	Immediate	User	Enables synchronized sequential scans.	
<i>synchronous_commit</i>	Session	Immediate	User	Sets immediate fsync at commit.	

continues on next page

Table 1 – continued from previous page

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>syn-chronous_standby_names</i>	Cluster	Reload	EPAS service account	Lists names of potential synchronous standbys.	
<i>syslog_facility</i>	Cluster	Reload	EPAS service account	Sets the syslog “facility” to be used when syslog enabled.	
<i>syslog_ident</i>	Cluster	Reload	EPAS service account	Sets the program name used to identify PostgreSQL messages in syslog.	
<i>tcp_keepalives_count</i>	Session	Immediate	User	Maximum number of TCP keepalive retransmits.	
<i>tcp_keepalives_idle</i>	Session	Immediate	User	Time between issuing TCP keepalives.	
<i>tcp_keepalives_interval</i>	Session	Immediate	User	Time between TCP keepalive retransmits.	
<i>temp_buffers</i>	Session	Immediate	User	Sets the maximum number of temporary buffers used by each session.	
<i>temp_file_limit</i>	Session	Immediate	Superuser	Limits the total size of all temporary files used by each session.	
<i>temp_tablespace</i>	Session	Immediate	User	Sets the tablespace(s) to use for temporary tables and sort files.	
<i>timed_statistics</i>	Session	Immediate	User	Enables the recording of timed statistics.	X
<i>timezone</i>	Session	Immediate	User	Sets the time zone for displaying and interpreting time stamps.	
<i>timezone_abbreviations</i>	Session	Immediate	User	Selects a file of time zone abbreviations.	
<i>trace_hints</i>	Session	Immediate	User	Emits debug info about hints being honored.	X
<i>trace_notify</i>	Session	Immediate	User	Generates debugging output for <i>LISTEN</i> and <i>NOTIFY</i> .	
<i>trace_recovery_messages</i>	Cluster	Reload	EPAS service account	Enables logging of recovery-related debugging information.	
<i>trace_sort</i>	Session	Immediate	User	Emits information about resource usage in sorting.	
<i>track_activities</i>	Session	Immediate	Superuser	Collects information about executing commands.	

continues on next page

Table 1 – continued from previous page

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>track_activity_query_size</i>	Cluster	Restart	EPAS service account	Sets the size reserved for <i>pg_stat_activity.current_query</i> , in bytes.	
<i>track_counts</i>	Session	Immediate	Superuser	Collects statistics on database activity.	
<i>track_functions</i>	Session	Immediate	Superuser	Collects function-level statistics on database activity.	
<i>track_io_timing</i>	Session	Immediate	Superuser	Collects timing statistics for database I/O activity.	
<i>transaction_deferrable</i>	Session	Immediate	User	Specifies whether to defer a read-only serializable transaction until it can be executed with no possible serialization failures.	
<i>transaction_isolation</i>	Session	Immediate	User	Sets the current transaction's isolation level.	
<i>transaction_read_only</i>	Session	Immediate	User	Sets the current transaction's read-only status.	
<i>transform_null_equals</i>	Session	Immediate	User	Treats " <i>expr=NULL</i> " as " <i>expr IS NULL</i> ".	
<i>unix_socket_directories</i>	Cluster	Restart	EPAS service account	Sets the directory where the Unix-domain socket is created.	
<i>unix_socket_group</i>	Cluster	Restart	EPAS service account	Sets the owning group of the Unix-domain socket.	
<i>unix_socket_permissions</i>	Cluster	Restart	EPAS service account	Sets the access permissions of the Unix-domain socket.	
<i>update_process_title</i>	Session	Immediate	Superuser	Updates the process title to show the active SQL command.	
<i>utl_encode.udcode_redword</i>	Session	Immediate	User	Allows decoding of Oracle-created unencoded data.	X
<i>utl_file.umask</i>	Session	Immediate	User	Umask used for files created through the <i>UTL_FILE</i> package.	X
<i>vacuum_cost_delay</i>	Session	Immediate	User	Vacuum cost delay in milliseconds.	
<i>vacuum_cost_limit</i>	Session	Immediate	User	Vacuum cost amount available before napping.	

continues on next page

Table 1 – continued from previous page

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>vacuum_cost_page_dirty</i>	Session	Immediate	User	Vacuum cost for a page dirtied by vacuum.	
<i>vacuum_cost_page_hit</i>	Session	Immediate	User	Vacuum cost for a page found in the buffer cache.	
<i>vacuum_cost_page_miss</i>	Session	Immediate	User	Vacuum cost for a page not found in the buffer cache.	
<i>vacuum_defer_cleanup_age</i>	Cluster	Reload	EPAS service count	Number of transactions by which <i>VACUUM</i> and <i>HOT</i> cleanup should be deferred, if any.	
<i>vacuum_freeze_min_age</i>	Session	Immediate	User	Minimum age at which <i>VACUUM</i> should freeze a table row.	
<i>vacuum_freeze_table_age</i>	Session	Immediate	User	Age at which <i>VACUUM</i> should scan whole table to freeze tuples.	
<i>vacuum_multixact_freeze_min_age</i>	Session	Immediate	User	Minimum age at which <i>VACUUM</i> should freeze a MultiXactId in a table row.	
<i>vacuum_multixact_freeze_table_age</i>	Session	Immediate	User	Multixact age at which <i>VACUUM</i> should scan whole table to freeze tuples.	
<i>wal_block_size</i>	Cluster	Preset	n/a	Shows the block size in the write ahead log.	
<i>wal_buffers</i>	Cluster	Restart	EPAS service count	Sets the number of disk-page buffers in shared memory for WAL.	
<i>wal_keep_segments</i>	Cluster	Reload	EPAS service count	Sets the number of WAL files held for standby servers.	
<i>wal_level</i>	Cluster	Restart	EPAS service count	Set the level of information written to the WAL.	
<i>wal_log_hints</i>	Cluster	Restart	EPAS service count	Writes full pages to WAL when first modified after a checkpoint, even for non-critical modifications.	
<i>wal_receiver_status_interval</i>	Cluster	Reload	EPAS service count	Sets the maximum interval between WAL receiver status reports to the primary.	
<i>wal_receiver_timeout</i>	Cluster	Reload	EPAS service count	Sets the maximum wait time to receive data from the primary.	

continues on next page

Table 1 – continued from previous page

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>wal_segment_size</i>	Cluster	Preset	n/a	Shows the number of pages per write ahead log segment.	
<i>wal_sender_timeout</i>	Cluster	Reload	EPAS service count	Sets the maximum time to wait for WAL replication.	
<i>wal_sync_method</i>	Cluster	Reload	EPAS service count	Selects the method used for forcing WAL updates to disk.	
<i>wal_writer_delay</i>	Cluster	Reload	EPAS service count	WAL writer sleep time between WAL flushes.	
<i>work_mem</i>	Session	Immediate	User	Sets the maximum memory to be used for query workspaces.	
<i>xloginsert_locks</i>	Cluster	Restart	EPAS service count	Sets the number of locks used for concurrent xlog insertions.	
<i>xmlbinary</i>	Session	Immediate	User	Sets how binary values are to be encoded in XML.	
<i>xmloption</i>	Session	Immediate	User	Sets whether XML data in implicit parsing and serialization operations is to be considered as documents or content fragments.	
<i>zero_damaged_pages</i>	Session	Immediate	Superuser	Continues processing past damaged page headers.	

3.1.3 Configuration parameters by functionality

The description of each group of parameters includes this list of attributes:

- ****Parametertype.**** Type of values the parameter can accept. See [:ref:`Setting configuration parameters<Setting configuration parameters>`](#) for a discussion of parameter type values.
- ****Defaultvalue.**** Default setting **if** a value isn't explicitly set in the configuration file.
- ****Range.**** Allowed range of values.
- ****Minimumscopeofeffect.**** Smallest scope for which a distinct setting can be made. Generally, the minimal scope of a distinct setting is either: - The entire cluster, meaning the setting is the same for all databases in the cluster and its sessions - `per session`, which means the setting might vary by role, database, or individual session. This attribute has the same meaning as the `Scope of effect` column in the table of [:ref:`Summary of configuration parameters<Summary of configuration parameters>`](#).
- ****Whenvaluechangestakeeffect.**** Least invasive action required to activate a change to a parameter's value. All parameter setting changes made in the configuration file can be put into effect by restarting the database server. However, certain parameters require a database server `restart`. Some parameter setting changes can be put into effect with a `reload` of the configuration file without stopping the database server. Finally, other parameter setting changes can be put into effect with some client-side action whose result is `immediate`. This attribute has the same meaning as the `When takes effect` column in the table of [:ref:`Summary of configuration parameters<Summary of configuration parameters>`](#).
- ****Requiredauthorizationtoactivate.**** The type of user authorization to activate a change to a parameter's setting. If a database server restart or a configuration file reload is required, then the user must be an EDB Postgres Advanced Server service account (`enterprisedb` or `postgres`, depending on the EDB Postgres Advanced Server compatibility installation mode). This attribute has the same meaning as the `Authorized user` column in the table of [:ref:`Summary of configuration parameters<Summary of configuration parameters>`](#).

top_performance_related_parameters resource_usage_memory resource_usage_edb_resource_manager
 query_tuning query_tuning_planner_method_configuration reporting_and_logging_what_to_log auditing_settings
 ccd_locale_and_formatting ccd_statement_behaviour ccd_other_defaults compatibility_options customized_options
 ungrouped audit_archiver

Top performance-related parameters

These configuration parameters have the most immediate impact on performance.

`shared_buffers` `work_mem` `maintenance_work_mem` `wal_buffers` `checkpoint_segments` `checkpoint_completion_target`
`checkpoint_timeout` `max_wal_size` `min_wal_size` `bgwriter_delay` `seq_page_cost` `random_page_cost`
`effective_cache_size` `synchronous_commit` `edb_max_spins_per_delay` `pg_prewarm.autoprewarm`
`pg_prewarm.autoprewarm_interval`

`shared_buffers`

Parameter type: Integer

Default value: 32MB

Range: 128kB to system dependent

Minimum scope of effect: Cluster

When value change takes effect: Restart

Required authorization to activate: EPAS service account

Sets the amount of memory the database server uses for shared memory buffers. The default is typically 32MB but might be less if your kernel settings don't support it, as determined during `initdb`. This setting must be at least 128kB. (Nondefault values of `BLCKSZ` change the minimum.) However, you usually need settings significantly higher than the minimum for good performance.

If you have a dedicated database server with 1GB or more of RAM, a reasonable starting value for `shared_buffers` is 25% of the memory in your system. For some workloads, even large settings for `shared_buffers` are effective. However, because EDB Postgres Advanced Server also relies on the operating system cache, allocating more than 40% of RAM to `shared_buffers` isn't likely to work better than a smaller amount.

On systems with less than 1GB of RAM, a smaller percentage of RAM is appropriate to leave space for the operating system. Fifteen percent of memory is more typical in these situations. Also, on Windows, large values for `shared_buffers` aren't as effective. You might have better results keeping the setting relatively low and using the operating system cache more instead. The useful range for `shared_buffers` on Windows systems is generally from 64MB to 512MB.

Increasing this parameter might cause EDB Postgres Advanced Server to request more System V shared memory than your operating system's default configuration allows. See [Shared Memory and Semaphores](#) in the PostgreSQL core documentation for information on how to adjust those parameters.

`work_mem`

Parameter type: Integer

Default value: 1MB

Range: 64kB to 2097151kB

Minimum scope of effect: Per session

When value change takes effect: Immediate

Required authorization to activate: Session user

Specifies the amount of memory for internal sort operations and hash tables to use before writing to temporary disk files.

For a complex query, several sort or hash operations might run in parallel. Each operation is allowed to use as much memory as this value specifies before it starts to write data into temporary files. Also, several running sessions might perform such operations concurrently. Therefore, the total memory used might be many times the value of `work_mem`. Keep this information in mind when choosing the value.

Sort operations are used for `ORDER BY`, `DISTINCT`, and merge joins. Hash tables are used in hash joins, hash-based aggregation, and hash-based processing of `IN` subqueries.

Reasonable values are typically between 4MB and 64MB, depending on the size of your machine, how many concurrent connections you expect (determined by `max_connections`), and the complexity of your queries.

`maintenance_work_mem`

Parametertype: Integer

Defaultvalue: 16MB

Range: 1024kB to 2097151kB

Minimumscopeofeffect: Per session

Whenvaluechangestakeeffect: Immediate

Requiredauthorizationtoactivate: Session user

Specifies the maximum amount of memory used by maintenance operations such as `VACUUM`, `CREATE INDEX`, and `ALTER TABLE ADD FOREIGN KEY`. It defaults to 16MB. Since a database session can execute only one of these operations at a time, and an installation normally doesn't have many of them running concurrently, it's safe to set this value significantly larger than `work_mem`. Larger settings might improve performance for vacuuming and for restoring database dumps.

When autovacuum runs, you can allocate up to `autovacuum_max_workers` times, so be careful not to set the default value too high.

A good rule of thumb is to set this to about 5% of system memory but not more than about 512MB. Larger values don't necessarily improve performance.

`wal_buffers`

Parametertype: Integer

Defaultvalue: 64kB

Range: 32kB to system dependent

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Restart

Requiredauthorizationtoactivate: EPAS service account

The amount of memory used in shared memory for WAL data. Because the data is written out to disk at every transaction commit, the setting must be large enough only to hold the amount of WAL data generated by one typical transaction.

Increasing this parameter might cause EDB Postgres Advanced Server to request more System V shared memory than your operating system's default configuration allows. See [Shared Memory and Semaphores](#) in the PostgreSQL core documentation for information on how to adjust those parameters.

Although even this very small setting doesn't always cause a problem, in some situations it can result in extra `fsync` calls and degrade overall system throughput. Increasing this value to about 1MB can address this problem. On very busy systems, you might need an even higher value, up to a maximum of about 16MB. Like `shared_buffers`, this

parameter increases EDB Postgres Advanced Server's initial shared memory allocation. If increasing it causes an EDB Postgres Advanced Server start failure, increase the operating system limit.

checkpoint_segments

Deprecated. This parameter isn't supported by EDB Postgres Advanced Server.

checkpoint_completion_target

Parametertype: Floating point

Defaultvalue: 0.5

Range: 0 to 1

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Reload

Requiredauthorizationtoactivate: EPAS service account

Specifies the target of checkpoint completion as a fraction of total time between checkpoints. This parameter spreads out the checkpoint writes while the system starts working toward the next checkpoint.

The default of 0.5 aims to finish the checkpoint writes when 50% of the next checkpoint is ready. A value of 0.9 aims to finish the checkpoint writes when 90% of the next checkpoint is done. This value throttles the checkpoint writes over a larger amount of time and avoids spikes of performance bottlenecking.

checkpoint_timeout

Parametertype: Integer

Defaultvalue: 5min

Range: 30s to 3600s

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Reload

Requiredauthorizationtoactivate: EPAS service account

Maximum time between automatic WAL checkpoints, in seconds. Increasing this parameter can increase the amount of time needed for crash recovery.

Increasing `checkpoint_timeout` to a larger value, such as 15 minutes, can reduce the I/O load on your system, especially when using large values for `shared_buffers`.

The downside of making these adjustments to the checkpoint parameters is that your system uses a modest amount of additional disk space and takes longer to recover in the event of a crash. However, for most users, this is worth it for a significant performance improvement.

max_wal_size

Parametertype: Integer

Defaultvalue: 1 GB

Range: 2 to 2147483647

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Reload

Requiredauthorizationtoactivate: EPAS service account

`max_wal_size` specifies the maximum size for the WAL to reach between automatic WAL checkpoints. This is a soft limit. WAL size can exceed `max_wal_size` under special circumstances, such as when under a heavy load, with a failing `archive_command`, or with a high `wal_keep_segments` setting.

Increasing this parameter can increase the amount of time needed for crash recovery. You can set this parameter in the `postgresql.conf` file or on the server command line.

min_wal_size

Parametertype: Integer

Defaultvalue: 80 MB

Range: 2 to 2147483647

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Reload

Requiredauthorizationtoactivate: EPAS service account

If WAL disk usage stays below the value specified by `min_wal_size`, old WAL files are recycled for future use at a checkpoint rather than removed. This feature ensures that enough WAL space is reserved to handle spikes in WAL usage, like when running large batch jobs. You can set this parameter in the `postgresql.conf` file or on the server command line.

bgwriter_delay

Parametertype: Integer

Defaultvalue: 200ms

Range: 10ms to 10000ms

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Reload

Requiredauthorizationtoactivate: EPAS service account

Specifies the delay between activity rounds for the background writer. In each round, the writer issues writes for some number of dirty buffers. (You can control the number of dirty buffers using the `bgwriter_lru_maxpages` and `bgwriter_lru_multiplier` parameters.) It then sleeps for `bgwriter_delay` milliseconds, and repeats.

On many systems, the effective resolution of sleep delays is 10ms. Setting `bgwriter_delay` to a value that isn't a multiple of 10 might have the same results as setting it to the next higher multiple of 10.

Typically, when tuning `bgwriter_delay`, you reduce it from its default value. This parameter is rarely increased. Saving on power consumption on a system with very low use is one case when you might increase the value.

`seq_page_cost`

Parametertype: Floating point

Defaultvalue: 1

Range: 0 to 1.79769e+308

Minimumscopeofeffect: Per session

Whenvaluechangestakeeffect: Immediate

Requiredauthorizationtoactivate: Session user

Sets the planner's estimate of the cost of a disk page fetch that's part of a series of sequential fetches. You can override this value for a particular tablespace by setting the tablespace parameter of the same name. See [ALTER TABLESPACE](#) in the PostgreSQL core documentation.

The default value assumes very little caching, so it's often a good idea to reduce it. Even if your database is significantly larger than physical memory, you might want to try setting this parameter to something lower than the default value of 1 to see if you get better query plans that way. If your database fits entirely in memory, you can lower this value much more, for example, to 0.1.

`random_page_cost`

Parametertype: Floating point

Defaultvalue: 4

Range: 0 to 1.79769e+308

Minimumscopeofeffect: Per session

Whenvaluechangestakeeffect: Immediate

Requiredauthorizationtoactivate: Session user

Sets the planner's estimate of the cost of a nonsequentially fetched disk page. You can override the default for a particular tablespace by setting the tablespace parameter of the same name. See [ALTER TABLESPACE](#) in the PostgreSQL core documentation.

Reducing this value relative to `seq_page_cost` causes the system to prefer index scans. Raising it makes index scans look relatively more expensive. You can raise or lower both values together to change the importance of disk I/O costs relative to CPU costs, which are described by the `cpu_tuple_cost` and `cpu_index_tuple_cost` parameters.

The default value assumes very little caching, so it's often a good idea to reduce it. Even if your database is significantly larger than physical memory, you might want to try setting this parameter to 2 (that is, lower than the default) to see whether you get better query plans that way. If your database fits entirely in memory, you can lower this value much more, for example, to 0.1.

Although the system allows it, never set `random_page_cost` less than `seq_page_cost`. However, setting them equal or very close to equal makes sense if the database fits mostly or entirely in memory, since in that case there's no penalty for touching pages out of sequence. Also, in a heavily cached database, lower both values relative to the CPU parameters, since the cost of fetching a page already in RAM is much smaller than it normally is.

effective_cache_size

Parameter type: Integer

Default value: 128MB

Range: 8kB to 17179869176kB

Minimum scope of effect: Per session

When value changes take effect: Immediate

Required authorization to activate: Session user

Sets the planner's assumption about the effective size of the disk cache that's available to a single query. This assumption is factored into estimates of the cost of using an index. A higher value makes it more likely index scans are used. A lower value makes it more likely sequential scans are used.

When setting this parameter, consider both EDB Postgres Advanced Server's shared buffers and the portion of the kernel's disk cache that are used for EDB Postgres Advanced Server data files. Also, consider the expected number of concurrent queries on different tables, since they have to share the available space.

This parameter doesn't affect the size of shared memory allocated by EDB Postgres Advanced Server, and it doesn't reserve kernel disk cache. Use it only for estimating.

If this parameter is set too low, the planner might decide not to use an index even when it's helpful to do so. Setting `effective_cache_size` to 50% of physical memory is a normal, conservative setting. A more aggressive setting is approximately 75% of physical memory.

synchronous_commit

Parameter type: Boolean

Default value: true

Range: {true | false}

Minimum scope of effect: Per session

When value changes take effect: Immediate

Required authorization to activate: Session user

Specifies whether a transaction commit waits for WAL records to be written to disk before the command returns a success indication to the client. The safe setting is on, which is the default. When off, there can be a delay between when success is reported to the client and when the transaction is really guaranteed to be safe against a server crash. (The maximum delay is three times `wal_writer_delay`.)

Unlike `fsync`, setting this parameter to off does not create any risk of database inconsistency. An operating system or database crash might result in some recent "allegedly committed" transactions being lost. However, the database state is the same as if those transactions aborted cleanly.

So, turning `synchronous_commit` off can be a useful alternative when performance is more important than exact certainty about the durability of a transaction. See [Asynchronous Commit](#) in the PostgreSQL core documentation for information.

You can change this parameter at any time. The behavior for any one transaction is determined by the setting in effect when it commits. It's therefore possible and useful to have some transactions commit synchronously and others asynchronously. For example, to make a single multistatement transaction commit asynchronously when the default is the opposite, issue `SET LOCAL synchronous_commit TO OFF` in the transaction.

edb_max_spins_per_delay

Parametertype: Integer

Defaultvalue: 1000

Range: 10 to 1000

Minimumscopeofeffect: Per cluster

Whenvaluechangestakeeffect: Restart

Requiredauthorizationtoactivate: EPAS service account

Specifies the maximum number of times that a session spins while waiting for a spin lock. If a lock isn't acquired, the session sleeps.

This parameter can be useful for systems that use non-uniform memory access (NUMA) architecture.

pg_prewarm.autoprewarm

Parametertype: Boolean

Defaultvalue: true

Range: {true | false}

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Restart

Requiredauthorizationtoactivate: EPAS service account

Controls whether the database server runs `autoprewarm`, a background worker process that dumps shared buffers to disk before a shutdown. It then *prewarms* the shared buffers the next time the server is started, meaning it loads blocks from the disk back into the buffer pool.

The advantage to this parameter is that it shortens the warmup times after the server restarts by loading the data that was dumped to disk before shutdown.

Set `pg_prewarm.autoprewarm` to on to enable the autoprewarm worker. Set it to off to disable `autoprewarm`.

Before you can use `autoprewarm`, you must add `$libdir/pg_prewarm` to the libraries listed in the `shared_preload_libraries` configuration parameter of the `postgresql.conf` file, as this example shows:

```
shared_preload_libraries = $libdir/dbms_pipe,$libdir/edb_gen,$libdir/dbms_aq,$libdir/pg_
↪prewarm
```

After modifying the `shared_preload_libraries` parameter, restart the database server. After the restart, the `autoprewarm` background worker launches immediately after the server reaches a consistent state.

The `autoprewarm` process starts loading blocks that were previously recorded in `$PGDATA/autoprewarm.blocks` until no free buffer space is left in the buffer pool. In this manner, any new blocks that were loaded either by the recovery process or by the querying clients aren't replaced.

Once the `autoprewarm` process finishes loading buffers from disk, it periodically dumps shared buffers to disk at the interval specified by the `pg_prewarm.autoprewarm_interval` parameter. At the next server restart, the `autoprewarm` process prewarms shared buffers with the blocks that were last dumped to disk.

See [`pg\_prewarm.autoprewarm\_interval`](#) for information on the `autoprewarm` background worker.

pg_prewarm.autoprewarm_interval**Parameter type:** Integer**Default value:** 300s**Range:** 0s to 2147483s**Minimum scope of effect:** Cluster**When value change takes effect:** Reload**Required authorization to activate:** EPAS service account

The minimum number of seconds after which the `autoprewarm` background worker dumps shared buffers to disk. If set to 0, shared buffers aren't dumped at regular intervals. They're dumped only when you shut down the server.

See the `pg_prewarm.autoprewarm` for information on the `autoprewarm` background worker.

Resource usage/memory

These configuration parameters control resource use pertaining to memory.

edb_dynatune**Parameter type:** Integer**Default value:** 0**Range:** 0 to 100**Minimum scope of effect:** Cluster**When value change takes effect:** Restart**Required authorization to activate:** EPAS service account

Determines how much of the host system's resources for the database server to use based on the host machine's total available resources and the intended use of the host machine.

When you first install EDB Postgres Advanced Server, you set `edb_dynatune` according to the use of the host machine on which it was installed, that is, development machine, mixed-use machine, or dedicated server. For most purposes, the database administrator doesn't need to adjust the various configuration parameters in the `postgresql.conf` file to improve performance.

You can set the `edb_dynatune` parameter to any integer value from 0 to 100. A value of 0 turns off the dynamic tuning feature, which leaves the database server resource use under the control of the other configuration parameters in the `postgresql.conf` file.

A low, non-zero value, for example, 1 to 33, dedicates the least amount of the host machine's resources to the database server. These values are suitable for a development machine where many other applications are being used.

A value in the range of 34 to 66 dedicates a moderate amount of resources to the database server. This setting might be used for a dedicated application server that has a fixed number of other applications running on the same machine as EDB Postgres Advanced Server.

The highest values of 67 to 100 dedicate most of the server's resources to the database server. Use settings in this range for a host machine that's dedicated to running EDB Postgres Advanced Server.

After you select a value for `edb_dynatune`, you can further fine-tune database server performance by adjusting the other configuration parameters in the `postgresql.conf` file. Any adjusted setting overrides the corresponding value

chosen by `edb_dynatune` . To change the value of a parameter, uncomment the configuration parameter, specify the desired value, and restart the database server.

`edb_dynatune_profile`

Parametertype: Enum

Defaultvalue: `oltp`

Range: {`oltp` | `reporting` | `mixed`}

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Restart

Requiredauthorizationtoactivate: EPAS service account

Controls tuning aspects based on the expected workload profile on the database server.

The following are the possible values:

- `oltp` . Recommended when the database server is processing heavy online transaction processing workloads.
- `reporting` . Recommended for database servers used for heavy data reporting.
- `mixed` . Recommended for servers that provide a mix of transaction processing and data reporting.

Resource usage/EDB Resource Manager

These configuration parameters control resource use through EDB Resource Manager.

`edb_max_resource_groups`

Parametertype: Integer

Defaultvalue: 16

Range: 0 to 65536

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Restart

Requiredauthorizationtoactivate: EPAS service account

Controls the maximum number of resource groups that EDB Resource Manager can use simultaneously. You can create more resource groups than the value specified by `edb_max_resource_groups` . However, the number of resource groups in active use by processes in these groups can't exceed this value.

Set this parameter large enough to handle the number of groups you expect to maintain.

edb_resource_group

Parametertype: String

Defaultvalue: none

Range: n/a

Minimumscopeofeffect: Per session

Whenvaluechangestakeeffect: Immediate

Requiredauthorizationtoactivate: Session user

The name of the resource group for EDB Resource Manager to control in the current session according to the group's resource type settings.

If you don't set this parameter, then the current session doesn't use EDB Resource Manager.

Query tuning

These configuration parameters are used for optimizer hints.

enable_hints

Parametertype: Boolean

Defaultvalue: true

Range: {true | false}

Minimumscopeofeffect: Per session

Whenvaluechangestakeeffect: Immediate

Requiredauthorizationtoactivate: Session user

Enables optimizer hints embedded in SQL commands. Optimizer hints are ignored when this parameter is off.

Query tuning/planner method configuration

These configuration parameters are used for planner method configuration.

edb_enable_pruning

Parametertype: Boolean

Defaultvalue: false

Range: {true | false}

Minimumscopeofeffect: Per session

Whenvaluechangestakeeffect: Immediate

Requiredauthorizationtoactivate: Session user

Allows the query planner to *early prune* partitioned tables. Early pruning means that the query planner can “prune” (that is, ignore) partitions that aren't searched in a query before generating query plans. This setting helps improve performance because it prevents generating query plans of partitions that aren't searched.

Conversely, *late pruning* means that the query planner prunes partitions after generating query plans for each partition. The `constraint_exclusion` configuration parameter controls late pruning.

The ability to early prune depends on the nature of the query in the `WHERE` clause. You can use early pruning in only simple queries with constraints like `WHERE column = literal`, for example, `WHERE deptno = 10`.

Don't use early pruning for more complex queries such as `WHERE column = expression`, for example, `WHERE deptno = 10 + 5`.

This parameter is deprecated from version 15 and later. Use `enable_partition_pruning` instead.

Reporting and logging/what to log

These configuration parameters control reporting and logging.

trace_hints

Parametertype: Boolean

Defaultvalue: false

Range: {true | false}

Minimumscopeofeffect: Per session

Whenvaluechangestakeeffect: Immediate

RequiredAauthorizationtoactivate: Session user

Use with the optimizer hints feature to provide more detailed information about whether the planner used a hint. Set the `client_min_messages` and `trace_hints` configuration parameters as follows:

```
SET client_min_messages TO info;
SET trace_hints TO true;
```

This example shows how the `SELECT` command with the `NO_INDEX` hint displays the added information produced when you set those configuration parameters:

```
EXPLAIN SELECT /*+ NO_INDEX(accounts accounts_pkey) */ * FROM accounts
WHERE aid = 100;

INFO: [HINTS] Index Scan of [accounts].[accounts_pkey] rejected because
of NO_INDEX hint.

INFO: [HINTS] Bitmap Heap Scan of [accounts].[accounts_pkey] rejected
because of NO_INDEX hint.
__OUTPUT__

          QUERY PLAN

-----
Seq Scan on accounts (cost=0.00..14461.10 rows=1 width=97)
  Filter: (aid = 100)
(2 rows)
```

edb_log_every_bulk_value

Parametertype: Boolean

Defaultvalue: false

Range: {true | false}

Minimumscopeofeffect: Per session

Whenvaluechangestakeeffect: Immediate

Requiredauthorizationtoactivate: Superuser

Bulk processing logs the resulting statements into both the EDB Postgres Advanced Server log file and the EDB Audit log file. However, logging every statement in bulk processing is costly. You can control the bulk processing statements that are logged with the `edb_log_every_bulk_value` configuration parameter.

When this parameter is set to `true`, every statement in bulk processing is logged. During bulk execution, when `edb_log_every_bulk_value` is set to `false`, a log message is recorded once per bulk processing along with the number of rows processed. The duration is emitted once per bulk processing.

Auditing settings

These configuration parameters are for use with the EDB Postgres Advanced Server database auditing feature.

`edb_audit` `edb_audit_directory` `edb_audit_filename` `edb_audit_rotation_day` `edb_audit_rotation_size`
`edb_audit_rotation_seconds` `edb_audit_connect` `edb_audit_disconnect` `edb_audit_statement` `edb_audit_tag`
`edb_audit_destination` `edb_log_every_bulk_value`

edb_audit

Parametertype: Enum

Defaultvalue: none

Range: {none | csv | xml}

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Reload

Requiredauthorizationtoactivate: EPAS service account

Enables or disables database auditing. The values `xml` or `csv` enable database auditing. These values determine the file format in which to capture auditing information. `none` disables database auditing.

edb_audit_directory

Parametertype: String

Defaultvalue: `edb_audit`

Range: n/a

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Reload

Requiredauthorizationtoactivate: EPAS service account

Specifies the directory where the audit log files are created. Specify either an absolute path or a path relative to the EDB Postgres Advanced Server data directory.

edb_audit_filename

Parametertype: String

Defaultvalue: audit-%Y%m%d_%H%M%S

Range: n/a

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Reload

Requiredauthorizationtoactivate: EPAS service account

Specifies the file name of the audit file where the auditing information is stored. The default file name is audit-%Y%m%d_%H%M%S. The escape sequences, such as %Y and %m, are replaced by the appropriate current values of the system date and time.

edb_audit_rotation_day

Parametertype: String

Defaultvalue: every

Range: {none | every | sun | mon | tue | wed | thu | fri | sat} ...

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Reload

Requiredauthorizationtoactivate: EPAS service account

Specifies the day of the week on which to rotate the audit files. Valid values are sun, mon, tue, wed, thu, fri, sat, every, and none. To disable rotation, set the value to none. To rotate the file every day, set the edb_audit_rotation_day value to every. To rotate the file on a specific day of the week, set the value to the desired day of the week.

edb_audit_rotation_size

Parametertype: Integer

Defaultvalue: 0MB

Range: 0MB to 5000MB

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Reload

Requiredauthorizationtoactivate: EPAS service account

Specifies a file size threshold in megabytes when file rotation occurs. If the parameter is commented out or set to 0, rotating the file based on size doesn't occur.

edb_audit_rotation_seconds**Parametertype:** Integer**Defaultvalue:** 0s**Range:** 0s to 2147483647s**Minimumscopeofeffect:** Cluster**Whenvaluechangestakeeffect:** Reload**Requiredauthorizationtoactivate:** EPAS service account

Specifies the rotation time in seconds when a new log file is created. To disable this feature, leave this parameter set to 0.

edb_audit_connect**Parametertype:** Enum**Defaultvalue:** failed**Range:** {none | failed | all}**Minimumscopeofeffect:** Cluster**Whenvaluechangestakeeffect:** Reload**Requiredauthorizationtoactivate:** EPAS service account

Enables auditing of database connection attempts by users. To disable auditing of all connection attempts, set `edb_audit_connect` to `none`. To audit all failed connection attempts, set the value to `failed`. To audit all connection attempts, set the value to `all`.

edb_audit_disconnect**Parametertype:** Enum**Defaultvalue:** none**Range:** {none | all}**Minimumscopeofeffect:** Cluster**Whenvaluechangestakeeffect:** Reload**Requiredauthorizationtoactivate:** EPAS service account

Enables auditing of database disconnections by connected users. To enable auditing of disconnections, set the value to `all`. To disable, set the value to `none`.

edb_audit_statement

Parametertype: String

Defaultvalue: ddl, error

Range: {none | ddl | dml | insert | update | delete | truncate | select | error | create | drop | alter | grant | revoke | rollback | set | all | { select | update | delete | insert }@groupname} ...

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Reload

Requiredauthorizationtoactivate: EPAS service account

Specifies auditing of different categories of SQL statements as well as statements related to specific SQL commands.

- To log errors, set the parameter value to `error`.
- To audit all DDL statements, such as `CREATE TABLE` and `ALTER TABLE`, set the parameter value to `ddl`.
- To audit specific types of DDL statements, the parameter values can include those specific SQL commands (`create`, `drop`, or `alter`). In addition, you can specify the object type following the command, such as `create table`, `create view`, and `drop role`.
- To audit all modification statements, such as `INSERT`, `UPDATE`, `DELETE`, or `TRUNCATE`, set `edb_audit_statement` to `dml`.
- To audit specific types of DML statements, the parameter values can include the specific SQL commands `insert`, `update`, `delete`, or `truncate`. Include parameter values `select`, `grant`, `revoke`, or `rollback` to audit statements regarding those SQL commands.
- To audit SET statements, include the parameter value `SET`.
- To audit every statement, set the value to `all`.
- To disable this feature, set the value to `none`.

The per-object level auditing audits the operations permitted by object privileges, such as `SELECT`, `UPDATE`, `DELETE`, and `INSERT` statements, including (@) and excluding (-) groups on a given table. To audit a specific type of object, specify the name of the object group to audit. The `edb_audit_statement` parameter can include the specific SQL commands (`select`, `update`, `delete`, or `insert`) associated with a group name with (@) include and (-) exclude symbol.

edb_audit_tag

Parametertype: String

Defaultvalue: none

Minimumscopeofeffect: Session

Whenvaluechangestakeeffect: Immediate

Requiredauthorizationtoactivate: User

Specifies a string value to include in the audit log when the `edb_audit` parameter is set to `csv` or `xml`.

edb_audit_destination

Parametertype: Enum

Defaultvalue: file

Range: {file | syslog}

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Reload

Requiredauthorizationtoactivate: EPAS service account

Specifies whether to record the audit log information in the directory as given by the `edb_audit_directory` parameter or to the directory and file managed by the `syslog` process. Set to `file` to use the directory specified by `edb_audit_directory`.

Set to `syslog` to use the `syslog` process and its location as configured in the `/etc/syslog.conf` file. The `syslog` setting is valid only for EDB Postgres Advanced Server running on a Linux host. It isn't supported on Windows systems.

Note: In recent Linux versions, `syslog` was replaced with `rsyslog`, and the configuration file is in `/etc/rsyslog.conf`.

edb_log_every_bulk_value

For information on `edb_log_every_bulk_value`, see [edb_log_every_bulk_value](#).

Client connection defaults/locale and formatting

These configuration parameters affect locale and formatting.

icu_short_form

Parametertype: String

Defaultvalue: none

Range: n/a

Minimumscopeofeffect: Database

Whenvaluechangestakeeffect: n/a

Requiredauthorizationtoactivate: n/a

Contains the default ICU short-form name assigned to a database or to the EDB Postgres Advanced Server instance. See [Unicode collation algorithm](#) for general information about the ICU short form and the Unicode collation algorithm.

Set this configuration parameter either when:

- Using the `CREATE DATABASE` command with the `ICU_SHORT_FORM` parameter. In this case, set the specified short-form name. It appears in the `icu_short_form` configuration parameter when connected to this database.
- Creating an EDB Postgres Advanced Server instance with the `initdb` command used with the `--icu_short_form` option. In this case, set the specified short-form name. It appears in the `icu_short_form` configuration parameter when connected to a database in that EDB Postgres Advanced Server instance. The database doesn't override it with its own `ICU_SHORT_FORM` parameter and a different short form.

Once you set this parameter, you can't change it.

Client connection defaults/statement behavior

These configuration parameters affect statement behavior.

default_heap_fillfactor

Parametertype: Integer

Defaultvalue: 100

Range: 10 to 100

Minimumscopeofeffect: Per session

Whenvaluechangestakeeffect: Immediate

Requiredauthorizationtoactivate: Session user

Sets the fill factor for a table when the `FILLFACTOR` storage parameter is omitted from a `CREATE TABLE` command.

The fill factor for a table is a percentage from 10 to 100, where 100 is complete packing. When you specify a smaller fill factor, `INSERT` operations pack table pages only to the indicated percentage. The remaining space on each page is reserved for updating rows on that page. This approach gives `UPDATE` a chance to place the updated copy of a row on the same page as the original, which is more efficient than placing it on a different page.

For a table whose entries are never updated, complete packing is the best choice. In heavily updated tables, use smaller fill factors.

edb_data_redaction

Parametertype: Boolean

Defaultvalue: true

Range: {true | false}

Minimumscopeofeffect: Per session

Whenvaluechangestakeeffect: Immediate

Requiredauthorizationtoactivate: Session user

Data redaction is the support of policies to limit the exposure of certain sensitive data to certain users by altering the displayed information.

When set to `TRUE`, the data redaction is applied to all users except for superusers and the table owner:

- Superusers and table owner bypass data redaction.
- All other users get the redaction policy applied and see the reformatted data.

When set to `FALSE`, the following occurs:

- Superusers and table owner still bypass data redaction.
- All other users get an error.

For information on data redaction, see *EDB Postgres Advanced Server Security Features*.

Client connection defaults/other defaults

These parameters set miscellaneous client connection defaults.

oracle_home

Parametertype: String

Defaultvalue: none

Range: n/a

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Restart

Requiredauthorizationtoactivate: EPAS service account

Before creating a link to an Oracle server, you must direct EDB Postgres Advanced Server to the correct Oracle home directory. Set the LD_LIBRARY_PATH environment variable on Linux or PATH on Windows to the lib directory of the Oracle client installation directory.

Alternatively, you can set the value of the oracle_home configuration parameter in the postgresql.conf file. The value specified in the oracle_home configuration parameter overrides the LD_LIBRARY_PATH environment variable in Linux and PATH environment variable in Windows.

Note: The *oracle_home* configuration parameter must provide the correct path to the Oracle client, that is, `OCI` library.

To set the oracle_home configuration parameter in the postgresql.conf file, add the following line:

```
oracle_home = <lib_directory>
```

<lib_directory> is the name of the oracle_home path to the Oracle client installation directory that contains libcIntsh.so in Linux and oci.dll in Windows.

After setting the oracle_home configuration parameter, you must restart the server for the changes to take effect. Restart the server:

- On Linux, using the systemctl command or pg_ctl services
- On Windows, from the Windows Services console

odbc_lib_path

Parametertype: String

Defaultvalue: none

Range: n/a

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Restart

Requiredauthorizationtoactivate: EPAS service account

If you're using an ODBC driver manager and if it's installed in a nonstandard location, specify the location by setting the odbc_lib_path configuration parameter in the postgresql.conf file:

```
odbc_lib_path = 'complete_path_to_libodbc.so'
```

The configuration file must include the complete pathname to the driver manager shared library, which is typically `libodbc.so`.

Compatibility options

These configuration parameters control various database compatibility features.

`edb_redwood_date`

Parametertype: Boolean

Defaultvalue: false

Range: {true | false}

Minimumscopeofeffect: Per session

Whenvaluechangestakeeffect: Immediate

Requiredauthorizationtoactivate: Session user

Translates DATE to TIMESTAMP when DATE appears as the data type of a column in the commands and the table definition is stored in the database. A time component is stored in the column along with the date.

If `edb_redwood_date` is set to FALSE, the column's data type in a CREATE TABLE or ALTER TABLE command remains as a native PostgreSQL DATE data type and is stored as such in the database. The PostgreSQL DATE data type stores only the date without a time component in the column.

Regardless of the setting of `edb_redwood_date`, when DATE appears as a data type in any other context, it's always internally translated to a TIMESTAMP. It can thus handle a time component if present. Examples of these contexts include:

- The data type of a variable in an SPL declaration section
- The data type of a formal parameter in an SPL procedure or SPL function
- The return type of an SPL function

`edb_redwood_greatest_least`

Parametertype: Boolean

Defaultvalue: true

Range: {true | false}

Minimumscopeofeffect: Per session

Whenvaluechangestakeeffect: Immediate

Requiredauthorizationtoactivate: Session user

The GREATEST function returns the parameter with the greatest value from its list of parameters. The LEAST function returns the parameter with the least value from its list of parameters.

When `edb_redwood_greatest_least` is set to TRUE, the GREATEST and LEAST functions return null when at least one of the parameters is null.

```
SET edb_redwood_greatest_least TO on;
```

```
SELECT GREATEST(1, 2, NULL, 3);
```

```
__OUTPUT__
```

```
greatest
```

```
- - - - -
```

```
(1 row)
```

When `edb_redwood_greatest_least` is set to `FALSE`, null parameters are ignored except when all parameters are null. In that case, the functions return null.

```
SET edb_redwood_greatest_least TO off;
```

```
SELECT GREATEST(1, 2, NULL, 3);
```

```
__OUTPUT__
```

```
greatest
```

```
- - - - -
```

```
3
```

```
(1 row)
```

```
SELECT GREATEST(NULL, NULL, NULL);
```

```
__OUTPUT__
```

```
greatest
```

```
- - - - -
```

```
(1 row)
```

edb_redwood_raw_names

Parameter type: Boolean

Default value: false

Range: {true | false}

Minimum scope of effect: Per session

When value changes take effect: Immediate

Required authorization to activate: Session user

When `edb_redwood_raw_names` is set to `FALSE`, database object names such as table names, column names, trigger names, program names, and user names appear in uppercase letters when viewed from Redwood catalogs (that is, system catalogs prefixed by `ALL_`, `DBA_`, or `USER_`). In addition, quotation marks enclose names that were created with enclosing quotation marks.

When `edb_redwood_raw_names` is set to `TRUE`, the database object names are displayed as they're stored in the PostgreSQL system catalogs when viewed from the Redwood catalogs. Names created without quotation marks around them appear in lower case as expected in PostgreSQL. Names created enclosed by quotation marks appear as they were created but without the quotation marks.

For example, the following user name is created, and then a session is started with that user:

```
CREATE USER reduser IDENTIFIED BY password;
edb=# \c - reduser
Password for user reduser:
You are now connected to database "edb" as user "reduser".
```

When connected to the database as `reduser`, the following tables are created:

```
CREATE TABLE all_lower (col INTEGER);
CREATE TABLE ALL_UPPER (COL INTEGER);
CREATE TABLE "Mixed_Case" ("Col" INTEGER);
```

When viewed from the Redwood catalog `USER_TABLES`, with `edb_redwood_raw_names` set to the default value `FALSE`, the names appear in upper case. The exception is the `Mixed_Case` name, which appears as created and also enclosed by quotation marks.

```
edb=> SELECT * FROM USER_TABLES;
__OUTPUT__
 schema_name | table_name | tablespace_name | status | temporary
-----+-----+-----+-----+-----
 REDUSER     | ALL_LOWER  |                  | VALID  | N
 REDUSER     | ALL_UPPER  |                  | VALID  | N
 REDUSER     | "Mixed_Case" |                  | VALID  | N
(3 rows)
```

When viewed with `edb_redwood_raw_names` set to `TRUE`, the names appear in lower case except for the `Mixed_Case` name, which appears as created but without quotation marks.

```
edb=> SET edb_redwood_raw_names TO true;
SET
edb=> SELECT * FROM USER_TABLES;
__OUTPUT__
 schema_name | table_name | tablespace_name | status | temporary
-----+-----+-----+-----+-----
 reduser     | all_lower  |                  | VALID  | N
 reduser     | all_upper  |                  | VALID  | N
 reduser     | Mixed_Case |                  | VALID  | N
(3 rows)
```

These names now match the case when viewed from the PostgreSQL `pg_tables` catalog:

```
edb=> SELECT schemaname, tablename, tableowner FROM pg_tables WHERE
tableowner = reduser;
__OUTPUT__
 schemaname | tablename | tableowner
-----+-----+-----
 reduser    | all_lower | reduser
 reduser    | all_upper | reduser
 reduser    | Mixed_Case | reduser
(3 rows)
```

edb_redwood_strings**Parameter type:** Boolean**Default value:** false**Range:** {true | false}**Minimum scope of effect:** Per session**When value changes take effect:** Immediate**Required authorization to activate:** Session user

If the `edb_redwood_strings` parameter is set to `TRUE`, when a string is concatenated with a null variable or null column, the result is the original string. If `edb_redwood_strings` is set to `FALSE`, the native PostgreSQL behavior is maintained, which is the concatenation of a string with a null variable or null column that gives a null result.

This example shows the difference. The sample application contains a table of employees. This table has a column named `comm` that's null for most employees. The following query is run with `edb_redwood_string` set to `FALSE`. Concatenating a null column with non-empty strings produces a final result of null, so only employees that have a commission appear in the query result. The output line for all other employees is null.

```
SET edb_redwood_strings TO off;

SELECT RPAD(ename,10) ||      || TO_CHAR(sal,99,999.99) ||      ||
TO_CHAR(comm,99,999.99) "EMPLOYEE COMPENSATION" FROM emp;
__OUTPUT__
  EMPLOYEE COMPENSATION
-----
ALLEN      1,600.00      300.00
WARD       1,250.00      500.00

MARTIN     1,250.00     1,400.00
TURNER     1,500.00          .00

(14 rows)
```

The following is the same query executed when `edb_redwood_strings` is set to `TRUE`. Here, the value of a null column is treated as an empty string. Concatenating an empty string with a non-empty string produces the non-empty string.

```
SET edb_redwood_strings TO on;

SELECT RPAD(ename,10) ||      || TO_CHAR(sal,99,999.99) ||      ||
TO_CHAR(comm,99,999.99) "EMPLOYEE COMPENSATION" FROM emp;
__OUTPUT__
  EMPLOYEE COMPENSATION
-----
SMITH 800.00
ALLEN 1,600.00 300.00
WARD 1,250.00 500.00
JONES 2,975.00
```

(continues on next page)

(continued from previous page)

```

MARTIN 1,250.00 1,400.00
BLAKE 2,850.00
CLARK 2,450.00
SCOTT 3,000.00
KING 5,000.00
TURNER 1,500.00 .00
ADAMS 1,100.00
JAMES 950.00
FORD 3,000.00
MILLER 1,300.00
(14 rows)

```

edb_stmt_level_tx**Parameter type:** Boolean**Default value:** false**Range:** {true | false}**Minimum scope of effect:** Per session**When value changes take effect:** Immediate**Required authorization to activate:** Session user

The term *statement-level transaction isolation* describes the behavior in which a runtime error occurs in a SQL command, and all the updates on the database caused by that single command are rolled back. For example, if a single UPDATE command successfully updates five rows, but an attempt to update a sixth row results in an exception, the updates to all six rows made by this UPDATE command are rolled back. The effects of prior SQL commands that haven't yet been committed or rolled back are pending until a COMMIT or ROLLBACK command is executed.

In EDB Postgres Advanced Server, if an exception occurs while executing a SQL command, all the updates on the database since the start of the transaction are rolled back. In addition, the transaction is left in an aborted state, and either a COMMIT or ROLLBACK command must be issued before another transaction can start.

If `edb_stmt_level_tx` is set to TRUE, then an exception doesn't roll back prior uncommitted database updates. If `edb_stmt_level_tx` is set to FALSE, then an exception rolls back uncommitted database updates.

Note: Use `edb_stmt_level_tx` set to TRUE only when necessary, as it can have a negative performance impact.

This example, run in PSQL, shows that when `edb_stmt_level_tx` is FALSE, the abort of the second INSERT command also rolls back the first INSERT command. In PSQL, you must issue the command `\set AUTOCOMMIT off`. Otherwise every statement commits automatically, which doesn't show the effect of `edb_stmt_level_tx`.

```

\set AUTOCOMMIT off
SET edb_stmt_level_tx TO off;

INSERT INTO emp (empno,ename,deptno) VALUES (9001, JONES, 40);
INSERT INTO emp (empno,ename,deptno) VALUES (9002, JONES, 00);
ERROR: insert or update on table "emp" violates foreign key constraint
"emp_ref_dept_fk"
DETAIL: Key (deptno)=(0) is not present in table "dept".

```

(continues on next page)

(continued from previous page)

```

COMMIT;
SELECT empno, ename, deptno FROM emp WHERE empno > 9000;
__OUTPUT__
empno | ename | deptno
- - - - - + - - - - - + - - - - -
(0 rows)

```

In this example, with `edb_stmt_level_tx` set to `TRUE`, the first `INSERT` command wasn't rolled back after the error on the second `INSERT` command. At this point, the first `INSERT` command can either be committed or rolled back.

```

\set AUTOCOMMIT off
SET edb_stmt_level_tx TO on;

INSERT INTO emp (empno,ename,deptno) VALUES (9001, JONES, 40);
INSERT INTO emp (empno,ename,deptno) VALUES (9002, JONES, 00);
ERROR: insert or update on table "emp" violates foreign key constraint
"emp_ref_dept_fk"
DETAIL: Key (deptno)=(0) is not present in table "dept"

SELECT empno, ename, deptno FROM emp WHERE empno > 9000;
__OUTPUT__
empno | ename | deptno
- - - - - + - - - - - + - - - - -
  9001 | JONES |    40
(1 row)

COMMIT;

```

If a `ROLLBACK` command is issued instead of the `COMMIT` command, the insert of employee number 9001 is rolled back as well.

db_dialect

Parametertype: Enum

Defaultvalue: postgres

Range: {postgres | redwood}

Minimumscopeofeffect: Per session

Whenvaluechangestakeeffect: Immediate

Requiredauthorizationtoactivate: Session user

In addition to the native PostgreSQL system catalog `pg_catalog`, EDB Postgres Advanced Server contains an extended catalog view. This is the `sys` catalog for the expanded catalog view. The `db_dialect` parameter controls the order in which these catalogs are searched for name resolution.

When set to `postgres`, the namespace precedence is `pg_catalog` and then `sys`, giving the PostgreSQL catalog the highest precedence. When set to `redwood`, the namespace precedence is `sys` and then `pg_catalog`, giving the expanded catalog views the highest precedence.

default_with_rowids

Parametertype: Boolean

Defaultvalue: false

Range: {true | false}

Minimumscopeofeffect: Per session

Whenvaluechangestakeeffect: Immediate

Requiredauthorizationtoactivate: Session user

When set to on, CREATE TABLE includes a ROWID column in newly created tables, which you can then reference in SQL commands. In earlier versions of EDB Postgres Advanced Server, ROWIDs were mapped to OIDs. With EDB Postgres Advanced Server version 12 and later, the ROWID is an autoincrementing value based on a sequence that starts with 1. It's assigned to each row of a table created with the ROWIDs option. By default, a unique index is created on a ROWID column.

The ALTER and DROP operations are restricted on a ROWID column.

To restore a database with ROWIDs with EDB Postgres Advanced Server 11 or an earlier version, you must perform the following:

- **pg_dump** : If a table includes OIDs then specify `--convert-oids-to-rowids` to dump a database. Otherwise, ignore the OIDs to continue table creation on EDB Postgres Advanced Server version 12 and later.
- **pg_upgrade** : Errors out. But if a table includes OIDs or ROWIDs, then you must perform the following: 1. Take a dump of the tables by specifying the `--convert-oids-to-rowids` option. 2. Drop the tables, and then perform the upgrade. 3. After the upgrade is successful, restore the dump into a new cluster that contains the dumped tables into a target database.

optimizer_mode

Parametertype: Enum

Defaultvalue: choose

Range: {choose | ALL_ROWS | FIRST_ROWS | FIRST_ROWS_10 | FIRST_ROWS_100 | FIRST_ROWS_1000}

Minimumscopeofeffect: Per session

Whenvaluechangestakeeffect: Immediate

Requiredauthorizationtoactivate: Session user

Sets the default optimization mode for analyzing optimizer hints.

The following table shows the possible values.

Hint	Description
<i>ALL_ROWS</i>	Optimizes for retrieval of all rows of the result set.
<i>CHOOSE</i>	Does no default optimization based on assumed number of rows to retrieve from the result set. This is the default.
<i>FIRST_ROWS</i>	Optimizes for retrieval of only the first row of the result set.
<i>FIRST_ROWS_10</i>	Optimizes for retrieval of the first 10 rows of the results set.
<i>FIRST_ROWS_100</i>	Optimizes for retrieval of the first 100 rows of the result set.
<i>FIRST_ROWS_1000</i>	Optimizes for retrieval of the first 1000 rows of the result set.

These optimization modes are based on the assumption that the client submitting the SQL command is interested in viewing only the first *n* rows of the result set and then abandons the rest of the result set. Resources allocated to the query are adjusted as such.

Customized options

`custom_variable_classes`

The `custom_variable_classes` parameter was deprecated in EDB Postgres Advanced Server 9.2. Parameters that previously depended on this parameter no longer require its support. In previous releases of EDB Postgres Advanced Server, `custom_variable_classes` was required by parameters not normally known to be added by add-on modules, such as procedural languages.

`dbms_alert.max_alerts`

Parametertype: Integer

Defaultvalue: 100

Range: 0 to 500

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Restart

Requiredauthorizationtoactivate: EPAS service account

Specifies the maximum number of concurrent alerts allowed on a system using the DBMS_ALERTS package.

`dbms_pipe.total_message_buffer`

Parametertype: Integer

Defaultvalue: 30 Kb

Range: 30 Kb to 256 Kb

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Restart

Requiredauthorizationtoactivate: EPAS service account

Specifies the total size of the buffer used for the DBMS_PIPE package.

index_advisor.enabled**Parametertype:** Boolean**Defaultvalue:** true**Range:** {true | false}**Minimumscopeofeffect:** Per session**Whenvaluechangestakeeffect:** Immediate**Requiredauthorizationtoactivate:** Session user

Temporarily suspends Index Advisor in an EDB-PSQL or PSQL session. To use this configuration parameter, the Index Advisor plugin `index_advisor` must be loaded in the EDB-PSQL or PSQL session.

You can load the Index Advisor plugin as follows:

```
$ psql -d edb -U enterprisedb
Password for user enterprisedb:
psql (14.0.0)
Type "help" for help.

edb=# LOAD index_advisor;
LOAD
```

Use the SET command to change the parameter setting to control whether Index Advisor generates an alternative query plan:

```
edb=# SET index_advisor.enabled TO off;
SET
edb=# EXPLAIN SELECT * FROM t WHERE a < 10000;
__OUTPUT__

          QUERY PLAN

-----
Seq Scan on t (cost=0.00..1693.00 rows=9864 width=8)
  Filter: (a < 10000)
(2 rows)
```

```
edb=# SET index_advisor.enabled TO on;
SET
edb=# EXPLAIN SELECT * FROM t WHERE a < 10000;
__OUTPUT__

          QUERY PLAN

-----
Seq Scan on t (cost=0.00..1693.00 rows=9864 width=8)
  Filter: (a < 10000)
Result (cost=0.00..327.88 rows=9864 width=8)
  One-Time Filter: ==[ HYPOTHETICAL PLAN ]===::text
  -> Index Scan using "<hypothetical-index>:1" on t (cost=0.00..327.88
rows=9864 width=8)
    Index Cond: (a < 10000)
(6 rows)
```

edb_sql_protect.enabled**Parameter type:** Boolean**Default value:** false**Range:** {true | false}**Minimum scope of effect:** Cluster**When value changes take effect:** Reload**Required authorization to activate:** EPAS service account

Controls whether SQL/Protect is actively monitoring protected roles by analyzing SQL statements issued by those roles and reacting according to the setting of `edb_sql_protect.level`. When you're ready to begin monitoring with SQL/Protect, set this parameter to `on`.

edb_sql_protect.level**Parameter type:** Enum**Default value:** passive**Range:** {learn | passive | active}**Minimum scope of effect:** Cluster**When value changes take effect:** Reload**Required authorization to activate:** EPAS service account

Sets the action taken by SQL/Protect when a SQL statement is issued by a protected role.

You can set this parameter to one of the following values to use learn mode, passive mode, or active mode:

- **learn**. Tracks the activities of protected roles and records the relations used by the roles. Use this value when initially configuring SQL/Protect so the expected behaviors of the protected applications are learned.
- **passive**. Issues warnings if protected roles are breaking the defined rules but doesn't stop any SQL statements from executing. This is the next step after SQL/Protect learns the expected behavior of the protected roles. This essentially behaves in intrusion detection mode. You can run it in production if you monitor it.
- **active**. Stops all invalid statements for a protected role. This behavior acts as a SQL firewall that prevents dangerous queries from running. This is particularly effective against early penetration testing when the attacker is trying to determine the vulnerability point and the type of database behind the application. Not only does SQL/Protect close those vulnerability points, it tracks the blocked queries. This behavior can alert administrators before the attacker finds an alternative method of penetrating the system.

If you're using SQL/Protect for the first time, set `edb_sql_protect.level` to `learn`.

edb_sql_protect.max_protected_relations

Parametertype: Integer

Defaultvalue: 1024

Range: 1 to 2147483647

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Restart

Requiredauthorizationtoactivate: EPAS service account

Sets the maximum number of relations that can be protected per role. The total number of protected relations for the server is the number of protected relations times the number of protected roles. Every protected relation consumes space in shared memory. The space for the maximum possible protected relations is reserved during database server startup.

If the server is started when `edb_sql_protect.max_protected_relations` is set to a value outside of the valid range (for example, a value of 2,147,483,648), then a warning message is logged in the database server log file:

```
2014-07-18 16:04:12 EDT WARNING: invalid value for parameter
"edb_sql_protect.max_protected_relations": "2147483648"
2014-07-18 16:04:12 EDT HINT: Value exceeds integer range.
```

The database server starts successfully but with `edb_sql_protect.max_protected_relations` set to the default value of 1024.

Although the upper range for the parameter is listed as the maximum value for an integer data type, the practical setting depends on how much shared memory is available and the parameter value used during database server startup.

As long as the space required can be reserved in shared memory, the value is acceptable. If the value is such that the space in shared memory can't be reserved, the database server startup fails with an error message like the following:

```
2014-07-18 15:22:17 EDT FATAL: could not map anonymous shared memory:
Cannot allocate memory
2014-07-18 15:22:17 EDT HINT: This error usually means that PostgreSQLs
request for a shared memory segment exceeded available memory, swap
space or huge pages. To reduce the request size (currently 2070118400
bytes), reduce PostgreSQLs shared memory usage, perhaps by reducing
shared_buffers or max_connections.
```

In this case, reduce the parameter value until you can start the database server successfully.

edb_sql_protect.max_protected_roles

Parametertype: Integer

Defaultvalue: 64

Range: 1 to 2147483647

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Restart

Requiredauthorizationtoactivate: EPAS service account

Sets the maximum number of roles that can be protected.

Every protected role consumes space in shared memory. The server reserves space for the number of protected roles times the number of protected relations (`edb_sql_protect.max_protected_relations`). The space for the maximum possible protected roles is reserved during database server startup.

If the database server is started when `edb_sql_protect.max_protected_roles` is set to a value outside of the valid range (for example, a value of 2,147,483,648), then a warning message is logged in the database server log file:

```
2014-07-18 16:04:12 EDT WARNING: invalid value for parameter
"edb_sql_protect.max_protected_roles": "2147483648"
2014-07-18 16:04:12 EDT HINT: Value exceeds integer range.
```

The database server starts successfully but with `edb_sql_protect.max_protected_roles` set to the default value of 64.

Although the upper range for the parameter is listed as the maximum value for an integer data type, the practical setting depends on how much shared memory is available and the parameter value used during database server startup.

As long as the space required can be reserved in shared memory, the value is acceptable. If the value is such that the space in shared memory can't be reserved, the database server startup fails with an error message such as the following:

```
2014-07-18 15:22:17 EDT FATAL: could not map anonymous shared memory:
Cannot allocate memory
2014-07-18 15:22:17 EDT HINT: This error usually means that PostgreSQLs
request for a shared memory segment exceeded available memory, swap
space or huge pages. To reduce the request size (currently 2070118400
bytes), reduce PostgreSQLs shared memory usage, perhaps by reducing
shared_buffers or max_connections.
```

In this cases, reduce the parameter value until you can start the database server successfully.

`edb_sql_protect.max_queries_to_save`

Parametertype: Integer

Defaultvalue: 5000

Range: 100 to 2147483647

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Restart

Requiredauthorizationtoactivate: EPAS service account

Sets the maximum number of offending queries to save in the view `edb_sql_protect_queries`.

Every saved query consumes space in shared memory. The space for the maximum possible queries that can be saved is reserved during database server startup.

If the database server is started when `edb_sql_protect.max_queries_to_save` is set to a value outside of the valid range (for example, a value of 10), then a warning message is logged in the database server log file:

```
2014-07-18 13:05:31 EDT WARNING: 10 is outside the valid range for
parameter "edb_sql_protect.max_queries_to_save" (100 .. 2147483647)
```

The database server starts successfully but with `edb_sql_protect.max_queries_to_save` set to the default value of 5000.

Although the upper range for the parameter is listed as the maximum value for an integer data type, the practical setting depends on how much shared memory is available and the parameter value used during database server startup.

As long as the space required can be reserved in shared memory, the value is acceptable. If the value is such that the space in shared memory can't be reserved, the database server startup fails with an error message like the following:

```
2014-07-18 15:22:17 EDT FATAL: could not map anonymous shared memory:
Cannot allocate memory
2014-07-18 15:22:17 EDT HINT: This error usually means that PostgreSQLs
request for a shared memory segment exceeded available memory, swap
space or huge pages. To reduce the request size (currently 2070118400
bytes), reduce PostgreSQLs shared memory usage, perhaps by reducing
shared_buffers or max_connections.
```

In this case, reduce the parameter value until you can start the database server successfully.

edb_wait_states.directory

Parametertype: String

Defaultvalue: edb_wait_states

Range: n/a

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Restart

Requiredauthorizationtoactivate: EPAS service account

Sets the directory path where the EDB wait states log files are stored. Use a full, absolute path, not a relative path. However, the default setting is `edb_wait_states`, which makes `$PGDATA/edb_wait_states` the default directory location. For information on EDB wait states, see `EDB wait states` under *Performance analysis and tuning*.

edb_wait_states.retention_period

Parametertype: Integer

Defaultvalue: 604800s

Range: 86400s to 2147483647s

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Reload

Requiredauthorizationtoactivate: EPAS service account

Sets the time to wait before deleting the log files for EDB wait states. The default is 604,800 seconds, which is 7 days. For information on EDB wait states, see `EDB wait dtates` under *Performance analysis and tuning*.

edb_wait_states.sampling_interval**Parametertype:** Integer**Defaultvalue:** 1s**Range:** 1s to 2147483647s**Minimumscopeofeffect:** Cluster**Whenvaluechangestakeeffect:** Reload**Requiredauthorizationtoactivate:** EPAS service account

Sets the timing interval between two sampling cycles for EDB wait states. The default setting is 1 second. For information on EDB wait states, see `EDB wait states` under *Performance analysis and tuning*.

edbldr.empty_csv_field**Parametertype:** Enum**Defaultvalue:** NULL**Range:** {NULL | empty_string | pgsql}**Minimumscopeofeffect:** Per session**Whenvaluechangestakeeffect:** Immediate**Requiredauthorizationtoactivate:** Session user

Use the `edbldr.empty_csv_field` parameter to specify how EDB*Loader treats an empty string. The table shows the valid values for the `edbldr.empty_csv_field` parameter.

Parameter setting	EDB*Loader behavior
<i>NULL</i>	An empty field is treated as <i>NULL</i> .
<i>empty_string</i>	An empty field is treated as a string of length zero.
<i>pgsql</i>	An empty field is treated as a <i>NULL</i> if it doesn't contain quotes and as an empty string if it contains quotes.

For more information about the `edbldr.empty_csv_field` parameter in EDB*Loader, see the Database Compatibility for Oracle Developers: Tools and Utilities.

utl_encode.uudecode_redwood**Parametertype:** Boolean**Defaultvalue:** false**Range:** {true | false}**Minimumscopeofeffect:** Per session**Whenvaluechangestakeeffect:** Immediate**Requiredauthorizationtoactivate:** Session user

When set to TRUE, EDB Postgres Advanced Server's `UTL_ENCODE.UUDECODE` function can decode uuencoded data that was created by the Oracle implementation of the `UTL_ENCODE.UUENCODE` function.

When set to FALSE, EDB Postgres Advanced Server's `UTL_ENCODE.UUDECODE` function can decode uuencoded data created by EDB Postgres Advanced Server's `UTL_ENCODE.UUENCODE` function.

utl_file.umask

Parametertype: String

Defaultvalue: 0077

Range: Octal digits for umask settings

Minimumscopeofeffect: Per session

Whenvaluechangestakeeffect: Immediate

Requiredauthorizationtoactivate: Session user

The `utl_file.umask` parameter sets the *file mode creation mask* in a manner similar to the Linux `umask` command. This is for use only within the EDB Postgres Advanced Server `UTL_FILE` package.

Note: The `utl_file.umask` parameter isn't supported on Windows systems.

The value specified for `utl_file.umask` is a three- or four-character octal string that's valid for the Linux `umask` command. The setting determines the permissions on files created by the `UTL_FILE` functions and procedures.

The following shows the results of the default `utl_file.umask` setting of 0077. All permissions are denied on users belonging to the `enterprisedb` group as well as all other users. Only the user `enterprisedb` has read and write permissions on the file.

```
- rw----- 1 enterprisedb enterprisedb 21 Jul 24 16:08 utlfile
```

Ungrouped

These configuration parameters apply only to EDB Postgres Advanced Server and are for a specific, limited purpose.

nls_length_semantics

Parametertype: Enum

Defaultvalue: byte

Range: {byte | char}

Minimumscopeofeffect: Per session

Whenvaluechangestakeeffect: Immediate

Requiredauthorizationtoactivate: Superuser

This parameter has no effect in EDB Postgres Advanced Server. For example, this form of the `ALTER SESSION` command is accepted in EDB Postgres Advanced Server without throwing a syntax error. However, it doesn't alter the session environment.

```
ALTER SESSION SET nls_length_semantics = char;
```

Note: Since setting this parameter has no effect on the server environment, it doesn't appear in the system view *pg_settings* .

query_rewrite_enabled

Parametertype: Enum

Defaultvalue: false

Range: {true | false | force}

Minimumscopeofeffect: Per session

Whenvaluechangestakeeffect: Immediate

Requiredauthorizationtoactivate: Session user

This parameter has no effect in EDB Postgres Advanced Server. For example, this form of the `ALTER SESSION` command is accepted in EDB Postgres Advanced Server without throwing a syntax error. However, it doesn't alter the session environment.

```
ALTER SESSION SET query_rewrite_enabled = force;
```

Note: Since setting this parameter has no effect on the server environment, it doesn't appear in the system view *pg_settings* .

query_rewrite_integrity

Parametertype: Enum

Defaultvalue: enforced

Range: {enforced | trusted | stale_tolerated}

Minimumscopeofeffect: Per session

Whenvaluechangestakeeffect: Immediate

Requiredauthorizationtoactivate: Superuser

This parameter has no effect in EDB Postgres Advanced Server. For example, this form of the `ALTER SESSION` command is accepted in EDB Postgres Advanced Server without throwing a syntax error. However, it doesn't alter the session environment.

```
ALTER SESSION SET query_rewrite_integrity = stale_tolerated;
```

Note: Since setting this parameter has no effect on the server environment, it doesn't appear in the system view *pg_settings* .

timed_statistics

Parametertype: Boolean

Defaultvalue: true

Range: {true | false}

Minimumscopeofeffect: Per session

Whenvaluechangestakeeffect: Immediate

Requiredauthorizationtoactivate: Session user

Controls collecting timing data for the Dynamic Runtime Instrumentation Tools Architecture (DRITA) feature. When set to on , timing data is collected.

Note: When EDB Postgres Advanced Server is installed, the *postgresql.conf* file contains an explicit entry that sets *timed_statistics* to *off* . If this entry is commented out and the configuration file is reloaded, timed statistics collection uses the default value, which is *on* .

Audit archiver

These configuration parameters are used by the EDB Postgres Advanced Server database audit archiving feature.

edb_audit_archiver

Parametertype: Enum

Defaultvalue: false

Range: {true | false}

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Restart

Requiredauthorizationtoactivate: EPAS service account

Enables or disables database audit archiving.

edb_audit_archiver_timeout

Parametertype: Integer

Defaultvalue: 300s

Range: 30s to 1d

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Reload

Requiredauthorizationtoactivate: EPAS service account

Enforces a timeout in seconds when a database attempts to archive a log file.

edb_audit_archiver_filename_prefix**Parametertype:** String**Defaultvalue:** audit-**Range:** n/a**Minimumscopeofeffect:** Cluster**Whenvaluechangestakeeffect:** Reload**Requiredauthorizationtoactivate:** EPAS service account

Specifies the file name of an audit log file that needs to be archived. The file name must align with the `edb_audit_filename` parameter. The audit files with `edb_audit_archiver_filename_prefix` in the `edb_audit_directory` are eligible for compression or expiration.

edb_audit_archiver_compress_time_limit**Parametertype:** Integer**Defaultvalue:** -1**Allowedvalue:** 0, -1, or any positive number value in seconds**Minimumscopeofeffect:** Cluster**Whenvaluechangestakeeffect:** Reload**Requiredauthorizationtoactivate:** EPAS service account

Specifies the time in seconds after which audit logs are eligible for compression. The possible values to set this parameter are:

- 0 . Compression starts as soon as the log file isn't a current file.
- -1 . Compression of the log file on a timely basis doesn't occur.

edb_audit_archiver_compress_size_limit**Parametertype:** Integer**Defaultvalue:** -1**Allowedvalue:** 0, -1, or any positive number value in megabytes**Minimumscopeofeffect:** Cluster**Whenvaluechangestakeeffect:** Reload**Requiredauthorizationtoactivate:** EPAS service account

Specifies a file size threshold in megabytes, after which audit logs are eligible for compression. If the parameter is set to -1, no compression of the log file occurs based on size.

edb_audit_archiver_compress_command

Parametertype: String

Defaultvalue: gzip %p

Range: n/a

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Reload

Requiredauthorizationtoactivate: EPAS service account

Specifies the command to execute compressing of the audit log files. The default value for `edb_audit_archiver_compress_command` is `gzip %p`. The `gzip` provides a standard method of compressing files. The `%p` in the string is replaced by the path name of the file to archive.

edb_audit_archiver_compress_suffix

Parametertype: String

Defaultvalue: .gz

Range: n/a

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Reload

Requiredauthorizationtoactivate: EPAS service account

Specifies the file name of an already compressed log file. The file name must align with `edb_audit_archiver_compress_command`. The default file name is `.gz`.

edb_audit_archiver_expire_time_limit

Parametertype: Integer

Defaultvalue: -1

Allowedvalue: 0, -1, or any positive number value in seconds

Minimumscopeofeffect: Cluster

Whenvaluechangestakeeffect: Reload

Requiredauthorizationtoactivate: EPAS service account

Specifies the time in seconds after which audit logs are eligible to expire. The possible values to set this parameter are:

- 0 . Expiration starts as soon as the log file isn't a current file.
- -1 . Expiration of the log file on a timely basis doesn't occur.

edb_audit_archiver_expire_size_limit**Parametertype:** Integer**Defaultvalue:** -1**Allowedvalue:** 0, -1, or any positive number value in megabytes**Minimumscopeofeffect:** Cluster**Whenvaluechangestakeeffect:** Reload**Requiredauthorizationtoactivate:** EPAS service account

Specifies a file size threshold in megabytes, after which audit logs are eligible to expire. If the parameter is set to -1, no expiration of a log file based on size occurs.

edb_audit_archiver_expire_command**Parametertype:** String**Defaultvalue:** ""**Range:** n/a**Minimumscopeofeffect:** Cluster**Whenvaluechangestakeeffect:** Reload**Requiredauthorizationtoactivate:** EPAS service account

Specifies the command to execute on an expired audit log file before removal.

edb_audit_archiver_sort_file**Parametertype:** String**Defaultvalue:** mtime**Range:** n/a**Minimumscopeofeffect:** Cluster**Whenvaluechangestakeeffect:** Reload**Requiredauthorizationtoactivate:** EPAS service account

Identifies the oldest log file to sort alphabetically or based on mtime .

- mtime sorts files based on file modification time.
- alphabetic sorts files alphabetically based on the file name.

3.2 Index Advisor

The Index Advisor utility helps determine the columns to index to improve performance in a given workload. Index Advisor considers B-tree (single-column or composite) index types. It doesn't identify other index types, that is, GIN, GiST, and Hash, that might improve performance. Index Advisor extension is installed with EDB Postgres Advanced Server.

Index Advisor works with EDB Postgres Advanced Server's query planner by creating *hypothetical indexes* that the query planner uses to calculate execution costs as if such indexes were available. Index Advisor identifies the indexes by analyzing SQL queries supplied in the workload.

You can use Index Advisor to analyze SQL queries in any of these ways:

- Invoke the Index Advisor utility program, supplying a text file containing the SQL queries that you want to analyze. Index Advisor generates a text file with `CREATE INDEX` statements for the recommended indexes.
- Provide queries at the EDB-PSQL command line that you want Index Advisor to analyze.
- Access Index Advisor through the Postgres Enterprise Manager (PEM) client. When accessed using the PEM client, Index Advisor works with SQL Profiler, providing indexing recommendations on code captured in SQL traces. For more information about using SQL Profiler and Index Advisor with PEM, see Using the Index Advisor in the PEM documentation.

Index Advisor attempts to make indexing recommendations on `INSERT`, `UPDATE`, `DELETE`, and `SELECT` statements. When invoking Index Advisor, you supply the workload in the form of either:

- If you're providing the command in an SQL file, a set of queries
- If you're specifying the SQL statement at the psql command line, an `EXPLAIN` statement

Index Advisor displays the query plan and estimated execution cost for the supplied query but doesn't execute the query.

During the analysis, Index Advisor compares the query execution costs with and without hypothetical indexes. If the execution cost using a hypothetical index is less than the execution cost without it:

- Both plans are reported in the `EXPLAIN` statement output.
- Metrics that quantify the improvement are calculated.
- Index Advisor generates the `CREATE INDEX` statement needed to create the index.

If no hypothetical index can be found that reduces the execution cost, Index Advisor displays only the original query plan output of the `EXPLAIN` statement.

Note: Index Advisor doesn't create indexes on the tables. Use the `CREATE INDEX` statements supplied by Index Advisor to add any recommended indexes to your tables.

An extension supplied with EDB Postgres Advanced Server creates the table in which Index Advisor stores the indexing recommendations generated by the analysis. The extension also creates a function and a view of the table to simplify retrieving and interpreting the results.

If you choose to forgo running the script, Index Advisor logs recommendations in a temporary table that's available only for the current Index Advisor session.

<code>index_advisor_components</code>	<code>index_advisor_configuration</code>	<code>using_index_advisor</code>	<code>review-</code>
<code>ing_the_index_advisor_recommendations</code>	<code>index_advisor_limitations</code>		

3.2.1 Index Advisor components

The Index Advisor shared library interacts with the query planner to make indexing recommendations.

On Windows, the EDB Postgres Advanced Server installer creates the shared library in the `libdir` subdirectory of your EDB Postgres Advanced Server home directory. The shared library is:

`index_advisor.dll`

For Linux, install the `edb-asxx-server-indexadvisor` RPM package, where `xx` is the EDB Postgres Advanced Server version number. The shared library is:

`index_advisor.so`

Only a superuser can load libraries in the `libdir` directory. A database administrator can allow a non-superuser to use Index Advisor by manually copying the Index Advisor file from the `libdir` directory into the `libdir/plugins` directory under your EDB Postgres Advanced Server home directory. Only allow a trusted non-superuser to have access to the plugin. This is an unsafe practice in a production environment.

The installer also creates the Index Advisor utility program and setup script:

`pg_advise_index`

A utility program that reads a user-supplied input file containing SQL queries and produces a text file containing `CREATE INDEX` statements. You can use these statements to create the indexes recommended by the Index Advisor. The `pg_advise_index` program is located in the `bin` subdirectory of the EDB Postgres Advanced Server home directory.

Note: `pg_advise_index` asks the backend process to load the `index_advisor` plugin first from `$libdir/plugins`. If not found, then it writes the error in the server log file and attempts to load from `$libdir`.

`index_advisor_log`

Index Advisor logs indexing recommendations in the `index_advisor_log` table. If Index Advisor doesn't find the `index_advisor_log` table in the user's search path, it stores any indexing recommendations in a temporary table of the same name. The temporary table exists only for rest of the current session.

`show_index_recommendations()`

A PL/pgSQL function that interprets and displays the recommendations made during a specific Index Advisor session, as identified by its backend process ID.

`index_recommendations`

Index Advisor creates the `index_recommendations` view based on information stored in the `index_advisor_log` table during a query analysis. The view produces output in the same format as the `show_index_recommendations()` function but contains Index Advisor recommendations for all stored sessions. The result set returned by the `show_index_recommendations()` function is limited to a specified session.

3.2.2 Index Advisor configuration

Index Advisor doesn't require configuration to generate recommendations that are available only for rest of the current session. To store the results of multiple sessions, you must create the `index_advisor_log` table, where EDB Postgres Advanced Server stores Index Advisor recommendations. To create the `index_advisor_log` table, create the extension `index_advisor`.

When selecting a storage schema for the Index Advisor table, function, and view, keep in mind that all users that invoke Index Advisor and query the result set must have `USAGE` privileges on the schema. The schema must be in the search path of all users that are interacting with Index Advisor.

1. Place the selected schema at the start of your `search_path` parameter. For example, suppose your search path is currently:

```
search_path=public, accounting
```

If you want to create the Index Advisor objects in a schema named `advisor`, use the command:

```
SET search_path = advisor, public, accounting;
```

2. Create the `index_advisor` extension, which creates the database objects. Log in as superuser and use the command:

```
edb=# create extension index_advisor ;  
__OUTPUT__  
CREATE EXTENSION
```

Note: If you're using Index Advisor in earlier version of EDB Postgres Advanced Server and upgrading to version 15, then use this command to create the `index_advisor` extension:

```
edb=# create extension index_advisor version 1.1;  
__OUTPUT__  
CREATE EXTENSION
```

This command creates the extension and links any of the old database objects to it.

3. Grant privileges on the `index_advisor_log` table to all Index Advisor users. This step isn't necessary if the Index Advisor user is a superuser or the owner of these database objects.
 - Grant `SELECT` and `INSERT` privileges on the `index_advisor_log` table to allow a user to invoke Index Advisor.
 - Grant `DELETE` privileges on the `index_advisor_log` table to allow the specified user to delete the table contents.
 - Grant `SELECT` privilege on the `index_recommendations` view.

This example shows creating Index Advisor database objects in a schema named `ia`. The schema is accessible to an Index Advisor user with user name `ia_user`.

```
$ edb-psql -d edb -U enterprisedb  
psql.bin (15.2.0, server 15.2.0)  
Type "help" for help.  
  
edb=# CREATE SCHEMA ia;  
CREATE SCHEMA  
edb=# SET search_path TO ia;  
SET  
edb=# CREATE EXTENSION index_advisor;  
CREATE EXTENSION  
edb=# GRANT USAGE ON SCHEMA ia TO ia_user;  
GRANT  
edb=# GRANT SELECT, INSERT, DELETE ON index_advisor_log TO ia_user;  
GRANT  
edb=# GRANT SELECT ON index_recommendations TO ia_user;  
GRANT
```

While using Index Advisor, the specified schema (`ia`) must be included in the `ia_user` `search_path` parameter.

3.2.3 Using Index Advisor

When you invoke Index Advisor, you must supply a workload. The workload is either a query specified at the command line or a file that contains a set of queries. These queries are executed by the `pg_advise_index()` function. After analyzing the workload, Index Advisor stores the result set in either a temporary table or in a permanent table. You can review the indexing recommendations generated by Index Advisor and use the `CREATE INDEX` statements generated by Index Advisor to create the recommended indexes.

Note: Don't run Index Advisor in read-only transactions.

The following examples assume that superuser `enterprisedb` is the Index Advisor user. The Index Advisor database objects were created in a schema in the `search_path` of superuser `enterprisedb`.

The examples use the table created with this statement:

```
CREATE TABLE t( a INT, b INT );
INSERT INTO t SELECT s, 99999 - s FROM generate_series(0,99999) AS s;
ANALYZE t;
```

The resulting table contains the following rows:

```
__OUTPUT__
a      | b
-----+-----
0      | 99999
1      | 99998
2      | 99997
3      | 99996
      .
      .
      .
99997  | 2
99998  | 1
99999  | 0
```

Using the `pg_advise_index` utility

When invoking the `pg_advise_index` utility, you must include the name of a file that contains the queries executed by `pg_advise_index`. The queries can be on the same line or on separate lines, but each query must be terminated by a semicolon. Queries in the file can't begin with the `EXPLAIN` keyword.

This example shows the contents of a sample `workload.sql` file:

```
SELECT * FROM t WHERE a = 500;
SELECT * FROM t WHERE b < 1000;
```

Run the `pg_advise_index` program:

```
$ pg_advise_index -d edb -h localhost -U enterprisedb -s 100M -o advisory.sql
workload.sql
poolsize = 102400 KB
load workload from file workload.sql
Analyzing queries .. done.
```

(continues on next page)

(continued from previous page)

```
size = 2184 KB, benefit = 1684.720000
size = 2184 KB, benefit = 1655.520000
/* 1. t(a): size=2184 KB, benefit=1684.72 */
/* 2. t(b): size=2184 KB, benefit=1655.52 */
/* Total size = 4368KB */
```

In the code sample, the `-d`, `-h`, and `-U` options are psql connection options.

`-s`

An optional parameter that limits the maximum size of the indexes recommended by Index Advisor. If Index Advisor doesn't return a result set, `-s` might be set too low.

`-o`

The recommended indexes are written to the file specified after the `-o` option.

The information displayed by the `pg_advise_index` program is logged in the `index_advisor_log` table. In response to the command shown in the example, Index Advisor writes the following `CREATE INDEX` statements to the `advisory.sql` output file:

```
create index idx_t_1 on t (a);
create index idx_t_2 on t (b);
```

You can create the recommended indexes at the psql command line with the `CREATE INDEX` statements in the file. Or you can create the indexes by executing the `advisory.sql` script.

```
$ edb-psql -d edb -h localhost -U enterprisedb -e -f advisory.sql
create index idx_t_1 on t (a);
CREATE INDEX
create index idx_t_2 on t (b);
CREATE INDEX
```

Note: `pg_advise_index` asks the backend process to load the `index_advisor` plugin first from `$libdir/plugins`. If not found, then it writes the error in the server log file and attempts to load from `$libdir`.

Using Index Advisor at the psql command line

You can use Index Advisor to analyze SQL statements entered at the `edb-psql` or `psql` command line.

To load the Index Advisor plugin and using Index Advisor:

1. Connect to the server with the `edb-psql` command line utility, and load the Index Advisor plugin:

```
$ edb-psql -d edb -U enterprisedb
...
edb=# LOAD index_advisor;
LOAD
```

2. Use the `edb-psql` command line to invoke each SQL command that you want Index Advisor to analyze. Index Advisor stores any recommendations for the queries in the `index_advisor_log` table. If the `index_advisor_log` table doesn't exist in the user's `search_path`, a temporary table is created with the same name. This temporary table exists only for the rest of the user's current session.

After you load the Index Advisor plugin, Index Advisor analyzes all SQL statements and logs any indexing recommendations for the rest of the session.

If you want Index Advisor to analyze a query and make indexing recommendations without executing the query, preface the SQL statement with the EXPLAIN keyword.

If you don't preface the statement with the EXPLAIN keyword, Index Advisor analyzes the statement while the statement executes. It writes the indexing recommendations to the `index_advisor_log` table for later review.

In this example, the EXPLAIN statement displays the normal query plan. It's followed by the query plan of the same query if the query were using the recommended hypothetical index:

```
edb=# EXPLAIN SELECT * FROM t WHERE a < 10000;
__OUTPUT__
          QUERY PLAN
-----
Seq Scan on t (cost=0.00..1693.00 rows=10105 width=8)
  Filter: (a < 10000)
Result (cost=0.00..337.10 rows=10105 width=8)
  One-Time Filter: ===[ HYPOTHETICAL PLAN ]===::text
  -> Index Scan using "<hypothetical-index>:1" on t
      (cost=0.00..337.10 rows=10105 width=8)
      Index Cond: (a < 10000)
(6 rows)
```

```
edb=# EXPLAIN SELECT * FROM t WHERE a = 100;
          QUERY PLAN
-----
Seq Scan on t (cost=0.00..1693.00 rows=1 width=8)
  Filter: (a = 100)
Result (cost=0.00..8.28 rows=1 width=8)
  One-Time Filter: ===[ HYPOTHETICAL PLAN ]===::text
  -> Index Scan using "<hypothetical-index>:3" on t
      (cost=0.00..8.28 rows=1 width=8)
      Index Cond: (a = 100)
(6 rows)
```

After loading the Index Advisor plugin, the default value of `index_advisor.enabled` is `on`. The Index Advisor plugin must be loaded to use a SET or SHOW command to display the current value of `index_advisor.enabled`.

You can use the `index_advisor.enabled` parameter to temporarily disable Index Advisor without interrupting the psql session:

```
edb=# SET index_advisor.enabled TO off;
SET
```

To enable Index Advisor, set the parameter to `on`:

```
edb=# SET index_advisor.enabled TO on;
SET
```

3.2.4 Reviewing the Index Advisor recommendations

You can review the index recommendations generated by Index Advisor in several ways. You can:

- Query the `index_advisor_log` table.
- Run the `show_index_recommendations` function.
- Query the `index_recommendations` view.

Using the `show_index_recommendations()` function

To review the recommendations of the Index Advisor utility using the `show_index_recommendations()` function, call the function, specifying the process ID of the session:

```
SELECT show_index_recommendations( pid );
```

Where `pid` is the process ID of the current session. If you don't know the process ID of your current session, passing a value of `NULL` also returns a result set for the current session.

This code fragment shows an example of a row in a result set:

```
edb=# SELECT show_index_recommendations(null);
__OUTPUT__
          show_index_recommendations
-----
create index idx_t_a on t(a);/* size: 2184 KB, benefit: 3040.62,
gain: 1.39222666981456 */
(1 row)
```

In the example, `create index idx_t_a on t(a)` is the SQL statement needed to create the index suggested by Index Advisor. Each row in the result set shows:

- The command required to create the recommended index.
- The maximum estimated size of the index.
- The calculated benefit of using the index.
- The estimated gain that results from implementing the index.

You can display the results of all Index Advisor sessions from the following view:

```
SELECT * FROM index_recommendations;
```

Querying the `index_advisor_log` table

Index Advisor stores indexing recommendations in a table named `index_advisor_log`. Each row in the `index_advisor_log` table contains the result of a query where Index Advisor determines it can recommend a hypothetical index to reduce the execution cost of that query.

Column	Type	Description
<i>reloid</i>	<i>oid</i>	OID of the base table for the index
<i>relname</i>	<i>name</i>	Name of the base table for the index

continues on next page

Table 4 – continued from previous page

Column	Type	Description
<i>attrs</i>	<i>integer[]</i>	Recommended index columns (identified by column number)
<i>benefit</i>	<i>real</i>	Calculated benefit of the index for this query
<i>index_size</i>	<i>integer</i>	Estimated index size in disk-pages
<i>backend_pid</i>	<i>integer</i>	Process ID of the process generating this recommendation
<i>timestamp</i>	<i>timestamp</i>	Date/time when the recommendation was generated

You can query the `index_advisor_log` table at the psql command line. This example shows the `index_advisor_log` table entries resulting from two Index Advisor sessions. Each session contains two queries and can be identified in the table by a different `backend_pid` value. For each session, Index Advisor generated two index recommendations.

```
edb=# SELECT * FROM index_advisor_log;
__OUTPUT__
 reloid | relname | attrs | benefit | index_size | backend_pid | timestamp
-----+-----+-----+-----+-----+-----+-----
-----
 16651  |  t      | {1}   | 1684.72 |    2184    |    3442    | 22-MAR-11
16:44:32.712638 -04:00
 16651  |  t      | {2}   | 1655.52 |    2184    |    3442    | 22-MAR-11
16:44:32.759436 -04:00
 16651  |  t      | {1}   | 1355.9  |    2184    |    3506    | 22-MAR-11
16:48:28.317016 -04:00
 16651  |  t      | {1}   | 1684.72 |    2184    |    3506    | 22-MAR-11
16:51:45.927906 -04:00
(4 rows)
```

Index Advisor added the first two rows to the table after analyzing the following two queries executed by the `pg_advise_index` utility:

```
SELECT * FROM t WHERE a = 500;
SELECT * FROM t WHERE b < 1000;
```

The value of 3442 in column `backend_pid` identifies these results as coming from the session with process ID 3442 .

The value of 1 in column `attrs` in the first row indicates that the hypothetical index is on the first column of the table (column a of table t).

The value of 2 in column `attrs` in the second row indicates that the hypothetical index is on the second column of the table (column b of table t).

Index Advisor added the last two rows to the table after analyzing the following two queries executed at the psql command line:

```
edb=# EXPLAIN SELECT * FROM t WHERE a < 10000;
__OUTPUT__
          QUERY PLAN
```

```
-----
-----
```

(continues on next page)

(continued from previous page)

```
Seq Scan on t (cost=0.00..1693.00 rows=10105 width=8)
  Filter: (a < 10000)
Result (cost=0.00..337.10 rows=10105 width=8)
  One-Time Filter: ===[ HYPOTHETICAL PLAN ]====:text
  -> Index Scan using "<hypothetical-index>:1" on t (cost=0.00..337.10
rows=10105 width=8)
      Index Cond: (a < 10000)
(6 rows)
```

```
edb=# EXPLAIN SELECT * FROM t WHERE a = 100;
__OUTPUT__

          QUERY PLAN
-----
Seq Scan on t (cost=0.00..1693.00 rows=1 width=8)
  Filter: (a = 100)
Result (cost=0.00..8.28 rows=1 width=8)
  One-Time Filter: ===[ HYPOTHETICAL PLAN ]====:text
  -> Index Scan using "<hypothetical-index>:3" on t (cost=0.00..8.28
rows=1 width=8)
      Index Cond: (a = 100)
(6 rows)
```

The values in the `benefit` column of the `index_advisor_log` table are calculated using the following formula:

$$\text{benefit} = (\text{normal execution cost}) - (\text{execution cost with hypothetical index})$$

The value of the `benefit` column for the last row of the `index_advisor_log` table shown in the example is calculated using the query plan for the following SQL statement:

```
EXPLAIN SELECT * FROM t WHERE a = 100;
```

The execution costs of the different execution plans are evaluated and compared:

$$\text{benefit} = (\text{Seq Scan on t cost}) - (\text{Index Scan using <hypothetical-index>})$$

The benefit is added to the table:

```
benefit = 1693.00 - 8.28
```

```
benefit = 1684.72
```

You can delete rows from the `index_advisor_log` table when you no longer need to review the results of the queries stored in the row.

Querying the index_recommendations view

The `index_recommendations` view contains the calculated metrics and the `CREATE INDEX` statements to create the recommended indexes for all sessions whose results are currently in the `index_advisor_log` table. You can display the results of all stored Index Advisor sessions by querying the `index_recommendations` view:

```
SELECT * FROM index_recommendations;
```

Using the example shown in [Querying the index_advisor_log table](#), the `index_recommendations` view displays the following:

```
edb=# SELECT * FROM index_recommendations;
__OUTPUT__
 backend_pid | show_index_recommendations
-----+-----
 3442      | create index idx_t_a on t(a);/* size: 2184 KB, benefit:
1684.72, gain: 0.771392654586624 */
 3442      | create index idx_t_b on t(b);/* size: 2184 KB, benefit:
1655.52, gain: 0.758021539820856 */
 3506      | create index idx_t_a on t(a);/* size: 2184 KB, benefit:
3040.62, gain: 1.39222666981456 */
(3 rows)
```

In each session, the results of all queries that benefit from the same recommended index are combined to produce one set of metrics per recommended index, reflected in the fields named `benefit` and `gain`.

The formulas for the fields are:

```
size = MAX(index size of all queries)
benefit = SUM(benefit of each query)
gain = SUM(benefit of each query) / MAX(index size of all queries)
```

So, for example, using the following query results from the process with a `backend_pid` of 3506 :

```
__OUTPUT__
 reloid | relname | attrs | benefit | index_size | backend_pid | timestamp
-----+-----+-----+-----+-----+-----+-----
 16651  | t       | {1}   | 1355.9  | 2184       | 3506       | 22-MAR-11
16:48:28.317016 -04:00
 16651  | t       | {1}   | 1684.72 | 2184       | 3506       | 22-MAR-11
16:51:45.927906 -04:00
```

The metrics displayed from the `index_recommendations` view for `backend_pid` 3506 are:

```
__OUTPUT__
 backend_pid | show_index_recommendations
-----+-----
 3506      | create index idx_t_a on t(a);/* size: 2184 KB, benefit:
3040.62, gain: 1.39222666981456 */
```

The metrics from the view are calculated like this:

```
benefit = (benefit from 1st query) + (benefit from 2nd query)
benefit = 1355.9 + 1684.72
benefit = 3040.62
```

As well as the following:

```
gain = ((benefit from 1st query) + (benefit from 2nd query)) / MAX(index
size of all queries)
gain = (1355.9 + 1684.72) / MAX(2184, 2184)
gain = 3040.62 / 2184
gain = 1.39223
```

The gain metric is useful when comparing the relative advantage of the different recommended indexes derived during a given session. The larger the gain value, the better the cost effectiveness derived from the index weighed against the possible disk space consumption of the index.

3.2.5 Index Advisor limitations

Index Advisor doesn't consider Index Only scans. It does consider Index scans when making recommendations.

Index Advisor ignores any computations found in the WHERE clause. Effectively, the index field in the recommendations isn't any kind of expression. The field is a simple column name.

Index Advisor doesn't consider inheritance when recommending hypothetical indexes. If a query references a parent table, Index Advisor doesn't make any index recommendations on child tables.

Suppose you're restoring a `pg_dump` backup file that includes the `index_advisor_log` table or any tables for which indexing recommendations were made and stored in the `index_advisor_log` table. Changes in object identifiers (OIDs) can result in "broken links" between the `index_advisor_log` table and the restored tables referenced by rows in the `index_advisor_log` table.

If you need to display the recommendations made prior to the backup, you can replace the old OIDs in the `reloid` column of the `index_advisor_log` table with the new OIDs of the referenced tables using the SQL UPDATE statement:

```
UPDATE index_advisor_log SET reloid = new_oid WHERE reloid = old_oid;
```

3.3 SQL Profiler

Inefficient SQL code is one of, if not the leading cause of database performance problems. The challenge for database administrators and developers is locating and then optimizing this code in large, complex systems.

- SQL Profiler* helps you locate and optimize poorly running SQL code.

Specific features and benefits of SQL Profiler include the following:

- On-demand traces. You can capture SQL traces at any time by manually setting up your parameters and starting the trace.
- Scheduled traces. For inconvenient times, you can also specify your trace parameters and schedule them to run at some later time.
- Save traces. Execute your traces and save them for later review.
- Trace filters. Selectively filter SQL captures by database and by user, or capture every SQL statement sent by all users against all databases.

- Trace output analyzer. A graphical table lets you quickly sort and filter queries by duration or statement, and a graphical or text based EXPLAIN plan lays out your query paths and joins.
- Index Advisor integration. Once you have found your slow queries and optimized them, you can also let the Index Advisor recommend the creation of underlying table indices to further improve performance.

The following describes the installation process.

Step1: Install SQL Profiler

SQL Profiler is installed by the EDB Postgres Advanced Server installer on Windows or from the `edb-asxx-server-sqlprofiler` RPM package on Linux where `xx` is the EDB Postgres Advanced Server version number.

Step2: Add the SQL Profiler library

Modify the `postgresql.conf` parameter file for the instance to include the SQL Profiler library in the `shared_preload_libraries` configuration parameter.

For Linux installations, the parameter value should include:

```
$libdir/sql-profiler
```

On Windows, the parameter value should include:

```
$libdir\sql-profiler.dll
```

Step3: Create the functions used by SQL Profiler

The SQL Profiler installation program places a SQL script (named `sql-profiler.sql`) in:

On Linux:

```
/usr/edb/as14/share/contrib/
```

On Windows:

```
C:\Program Files\edb\as14\share\contrib\
```

Use the `psql` command line interface to run the `sql-profiler.sql` script in the database specified as the Maintenance Database on the server you wish to profile. If you are using EDB Postgres Advanced Server, the default maintenance database is named `edb`. If you are using a PostgreSQL instance, the default maintenance database is named `postgres`.

The following command uses the `psql` command line to invoke the `sql-profiler.sql` script on a Linux system:

```
$ /usr/edb/as14/bin/psql -U user_name database_name <
/usr/edb/as14/share/contrib/sql-profiler.sql
```

Step4: Stop and restart the server for the changes to take effect.

After configuring SQL Profiler, it is ready to use with all databases that reside on the server. You can take advantage of SQL Profiler functionality with EDB Postgres Enterprise Manager; for more information about Postgres Enterprise Manager, visit the EDB website at:

<https://www.enterprisedb.com/docs>

3.3.1 Troubleshooting

If (after performing an upgrade to a newer version of SQL Profiler) you encounter an error that contains the following text:

An error has occurred:
ERROR: function return row and query-specified return row do not match.
DETAIL: Returned row contains 11 attributes, but the query expects 10.

To correct this error, you must replace the existing query set with a new query set. First, uninstall SQL Profiler by invoking the `uninstall-sql-profiler.sql` script, and then reinstall SQL Profiler by invoking the `sql-profiler.sql` script.

3.4 pgsnmpd

Note: ‘pgsnmpd’ is deprecated as of EDB Postgres Advanced Server v14. It isn’t included in EDB Postgres Advanced Server v15 and later.

`pgsnmpd` is an SNMP agent that can return hierarchical information about the current state of EDB Postgres Advanced Server on a Linux system. `pgsnmpd` is distributed and installed using the `edb-asxx-pgsnmpd` RPM package where `xx` is the EDB Postgres Advanced Server version number. The `pgsnmpd` agent can operate as a stand-alone SNMP agent, as a pass-through sub-agent, or as an AgentX sub-agent.

After installing EDB Postgres Advanced Server, you must update the `LD_LIBRARY_PATH` variable. Use the command:

```
$ export LD_LIBRARY_PATH=/usr/edb/as14/lib:$LD_LIBRARY_PATH
```

This command does not persistently alter the value of `LD_LIBRARY_PATH`. Consult the documentation for your distribution of Linux for information about persistently setting the value of `LD_LIBRARY_PATH`.

The examples that follow demonstrate the simplest usage of `pgsnmpd`, implementing read only access. `pgsnmpd` is based on the `net-snmp` library; for more information about `net-snmp`, visit:

<http://net-snmp.sourceforge.net/>

3.4.1 Configuring pgsnmpd

The `pgsnmpd` configuration file is named `snmpd.conf`. For information about the directives that you can specify in the configuration file, review the `snmpd.conf` man page (`man snmpd.conf`).

You can create the configuration file by hand, or you can use the `snmpconf` perl script to create the configuration file. The perl script is distributed with `net-snmp` package.

`net-snmp` is an open-source package available from:

<http://www.net-snmp.org/>

To use the `snmpconf` configuration file wizard, download and install `net-snmp`. When the installation completes, open a command line and enter:

```
snmpconf
```

When the configuration file wizard opens, it may prompt you to read in an existing configuration file. Enter `none` to generate a new configuration file (not based on a previously existing configuration file).

`snmpconf` is a menu-driven wizard. Select menu item 1: `snmpd.conf` to start the configuration wizard. As you select each top-level menu option displayed by `snmpconf`, the wizard walks through a series of questions, prompting you for information required to build the configuration file. When you have provided information in each of the category relevant to your system, enter `Finished` to generate a configuration file named `snmpd.conf`. Copy the file to:

```
/usr/edb/as14/share/
```

3.4.2 Setting the listener address

By default, `pgsnmpd` listens on port 161. If the listener port is already being used by another service, you may receive the following error:

```
Error opening specified endpoint "udp:161".
```

You can specify an alternate listener port by adding the following line to your `snmpd.conf` file:

```
agentaddress $host_address:2000
```

The example instructs `pgsnmpd` to listen on UDP port 2000, where `$host_address` is the IP address of the server (e.g., `127.0.0.1`).

Before invoking `pgsnmpd`, you must create the `pgsnmpd` schema and set the search path to create tables in the `pgsnmpd` schema using the following commands:

```
CREATE SCHEMA pgsnmpd;
SET search_path = pgsnmpd;
\i pgsnmpd.sql
```

3.4.3 Invoking pgsnmpd

Ensure that an instance of EDB Postgres Advanced Server is up and running (`pgsnmpd` connects to this server). Open a command line and assume superuser privileges, before invoking `pgsnmpd` with a command that takes the following form:

```
POSTGRES_INSTALL_HOME/bin/pgsnmpd -b
  -c POSTGRES_INSTALL_HOME/share/snmpd.conf
  -C "user=enterprisedb dbname=edb password=safe_password
      port=5444"
```

Where `POSTGRES_INSTALL_HOME` specifies the EDB Postgres Advanced Server installation directory.

Include the `-b` option to specify that `pgsnmpd` should run in the background.

Include the `-c` option, specifying the path and name of the `pgsnmpd` configuration file.

Include connection information for your installation of EDB Postgres Advanced Server (in the form of a `libpq` connection string) after the `-C` option.

3.4.4 Viewing pgsnmpd help

Include the `--help` option when invoking the `pgsnmpd` utility to view other `pgsnmpd` command line options:

```
pgsnmpd --help
Version PGSQL-SNMP-Ver1.0
usage: pgsnmpd [-s] [-b] [-c FILE ] [-x address ] [-g] [-C "Connect String"]
  -s : run as AgentX sub-agent of an existing snmpd process
  -b : run in the background
  -c : configuration file name
  -g : use syslog
  -C : libpq connection string
  -x : address:port of a network interface
  -V : display version strings
```

3.4.5 Requesting information from pgsnmpd

You can use `net-snmp` commands to query the `pgsnmpd` service. For example:

```
snmpgetnext -v 2c -c public localhost .1.3.6.1.4.1.5432.1.4.2.1.1.0
```

In the above example:

`-v 2c` option instructs the `snmpgetnext` client to send the request in SNMP version 2c format.

`-c public` specifies the community name.

`localhost` indicates the host machine running the `pgsnmpd` server.

`.1.3.6.1.4.1.5432.1.4.2.1.1.0` is the identity of the requested object. To see a list of all databases, increment the last digit by one (e.g. `.1.1`, `.1.2`, `.1.3` etc.).

The encodings required to query any given object are defined in the MIB (Management Information Base). An SNMP client can monitor a variety of servers; the server type determines the information exposed by a given server. Each SNMP server describes the exposed data in the form of a MIB (Management information base). By default, `pgsnmpd` searches for MIB's in the following locations:

```
/usr/share/snmp/mibs
$HOME/.snmp/mibs
```

3.5 EDB audit logging

EDB Postgres Advanced Server allows database and security administrators, auditors, and operators to track and analyze database activities using EDB *audit logging*. EDB audit logging generates audit log files, which contain all of the relevant information. You can configure the audit logs to record information such as:

- When a role establishes a connection to an EDB Postgres Advanced Server database
- The database objects a role creates, modifies, or deletes when connected to EDB Postgres Advanced Server
- When any failed authentication attempts occur

The parameters specified in the configuration files `postgresql.conf` or `postgresql.auto.conf` control the information included in the audit logs.

audit_logging_configuration_parameters selecting_sql_statements_to_audit enabling_audit_logging audit_log_file using_error_codes_to_filter_audit_logs using_command_tags_to_filter_audit_logs redacting_passwords_from_audit_logs audit_log_archiving object_auditing

3.5.1 Audit logging configuration parameters

Use the following configuration parameters to control database auditing. See *Summary of configuration parameters* to determine if a change to the configuration parameter:

- Takes effect immediately
- Requires reloading the configuration
- Requires restarting the database server

`edb_audit`

Enables or disables database auditing. The values `xml` or `csv` enable database auditing. These values represent the file format in which to capture auditing information. `none` disables database auditing and is the default.

Note: Set the `logging_collector` parameter to `on` to enable the `edb_audit` parameter.

`edb_audit_archiver`

Enables or disables database audit archiving.

`edb_audit_archiver_timeout`

Enforces a timeout in seconds when a database attempts to archive a log file. The valid range is 30 seconds to one day.

`edb_audit_archiver_filename_prefix`

Specifies the file name of an audit log file that needs to be archived. The file name must align with the `edb_audit_filename` parameter. The default value for `edb_audit_archiver_filename_prefix` is `audit-`. The audit files with `edb_audit_archiver_filename_prefix` in the `edb_audit_directory` are eligible for compression and expiration.

`edb_audit_archiver_compress_time_limit`

Specifies the time in seconds after which audit logs are eligible for compression. The possible values are: `-0` — Compression starts as soon as the log file isn't a current file. `-1` — Compression of the log file on a timely basis doesn't occur.

`edb_audit_archiver_compress_size_limit`

Specifies a file size threshold, in megabytes, after which audit logs are eligible for compression. If the parameter is set to `-1`, compression of the log file on a size basis doesn't occur.

`edb_audit_archiver_compress_command`

Specifies the command to execute compression of the audit log files. The default value for `edb_audit_archiver_compress_command` is `gzip %p`. The `gzip` provides a standard method of compressing files. The `%p` in the string is replaced with the path name of the file to archive.

`edb_audit_archiver_compress_suffix`

Specifies the file name of an already compressed log file. The file name must align with `edb_audit_archiver_compress_command`. The default file name is `.gz`.

`edb_audit_archiver_expire_time_limit`

Specifies the time, in seconds, after which audit logs are eligible to expire. The possible values to set this parameter are: - 0 — Expiration starts as soon as the log file isn't a current file. - -1 — Expiration of the log file on a timely basis doesn't occur.

`edb_audit_archiver_expire_size_limit`

Specifies a file size threshold in megabytes, after which audit logs are eligible to expire. If the parameter is set to -1, the expiration of a log file on the size basis doesn't occur.

`edb_audit_archiver_expire_command`

Specifies the command to execute on an expired audit log file before removal.

`edb_audit_archiver_sort_file`

Identifies the oldest log file to sort alphabetically or based on `mtime`. `mtime` indicates sorting of files based on file modification time. `alphabetic` indicates sorting of files alphabetically based on the file name.

`edb_audit_directory`

Specifies the directory where the log files are created. The path of the directory can be relative to the data folder or absolute. The default is the `PGDATA/edb_audit` directory.

`edb_audit_filename`

Specifies the file name of the audit file where the auditing information are stored. The default file name is `audit-%Y%m%d_%H%M%S`. The escape sequences `%Y`, `%m`, and so on are replaced by the appropriate current values according to the system date and time.

`edb_audit_rotation_day`

Specifies the day of the week on which to rotate the audit files. Valid values are `sun`, `mon`, `tue`, `wed`, `thu`, `fri`, `sat`, `every`, and `none`. To disable rotation, set the value to `none`. To rotate the file every day, set the `edb_audit_rotation_day` value to `every`. To rotate the file on a specific day of the week, set the value to the desired day of the week. `every` is the default value.

`edb_audit_rotation_size`

Specifies a file size threshold, in megabytes, when file rotation is forced to occur. The default value is 0 MB. If the parameter is commented out or set to 0, rotation of the file on a size basis doesn't occur.

`edb_audit_rotation_seconds`

Specifies the rotation time in seconds to create a log file. To disable this feature, set this parameter to 0, which is the default.

`edb_audit_connect`

Enables auditing of database connection attempts by users. To disable auditing of all connection attempts, set `edb_audit_connect` to `none`. To audit all connection attempts, set the value to `all`. To audit all failed connection attempts, set the value to `failed`, which is the default.

`edb_audit_disconnect`

Enables auditing of database disconnections by connected users. To enable auditing of disconnections, set the value to `all`. To disable, set the value to `none`, which is the default.

`edb_audit_statement`

Specifies auditing of different categories of SQL statements. You can specify various combinations of the following values: `none`, `dml`, `insert`, `update`, `delete`, `truncate`, `select`, `error`, `rollback`, `ddl`, `create`, `drop`, `alter`, `grant`, `revoke`, `set`, `all`, and `{ select | update | delete | insert }@groupname`. The default is `ddl` and `error`. See [Selecting SQL statements to audit](#) for information about setting this parameter.

`edb_audit_tag`

Specifies a string value to include in the audit log file for each entry as a tracking tag.

`edb_log_every_bulk_value`

Bulk processing logs the resulting statements into both the EDB Postgres Advanced Server log file and the EDB Audit log file. However, logging every statement in bulk processing is costly. You can control this with the `edb_log_every_bulk_value` configuration parameter. When set to `true`, every statement in bulk processing is logged. During bulk execution, when `edb_log_every_bulk_value` is set to `false`, a log message is recorded once per bulk processing along with the number of rows processed. In addition, the duration is emitted once per bulk processing. Default is `false`.

`edb_audit_destination`

Specifies whether to record the audit log information in the directory as given by the `edb_audit_directory` parameter or to the directory and file managed by the `syslog` process. Set to `file` to use the directory specified by `edb_audit_directory`, which is the default setting.

Set to `syslog` to use the `syslog` process and its location as configured in the `/etc/syslog.conf` file. The `syslog` setting is valid for EDB Postgres Advanced Server running on a Linux host and isn't supported on Windows systems. .. Note::

In recent Linux versions, `syslog` was replaced with `rsyslog`, and the configuration file is in `/etc/rsyslog.conf`.

Note: EDB Postgres Advanced Server allows administrative users associated with administrative privileges to audit statements by any user, group, or role. By auditing specific users, you can minimize the number of audit records generated. For information, see the examples under *Selecting SQL statements to audit*.

3.5.2 Selecting SQL statements to audit

The `edb_audit_statement` permits inclusion of one or more comma-separated values to control the SQL statements to audit. The following is the general format:

```
edb_audit_statement = value_1[, value_2]...
```

The comma-separated values can include or omit space characters following the comma. You can specify the values in any combination of lowercase or uppercase letters.

The basic parameter values are the following:

- `all` — Audit and log every statement including any error messages on statements.
- `none` — Disable all auditing and logging. A value of `none` overrides any other value included in the list.
- `ddl` — Audit all data definition language (DDL) statements (`CREATE`, `ALTER`, and `DROP`) as well as `GRANT` and `REVOKE` data control language (DCL) statements.
- `dml` — Audit all data manipulation language (DML) statements (`INSERT`, `UPDATE`, `DELETE`, and `TRUNCATE`).
- `select` — Audit `SELECT` statements.
- `set` — Audit `SET` statements.
- `rollback` — Audit `ROLLBACK` statements.
- `error` — Log all error messages that occur. Otherwise, no messages about errors that occur on SQL statements related to any of the other preceding parameter values are logged except when `all` is used.
- `{ select | update | delete | insert }@groupname` — Selectively audit objects for specific DML statements (`SELECT`, `UPDATE`, `DELETE`, and `INSERT`) including (`@`) and excluding (`-`) groups on a given table. For more information, see *Object auditing*.

DDL and DCL statements describes additional parameter values for selecting particular DDL or DCL statements for auditing. *DML statements* describes additional parameter values for selecting particular DML statements for auditing.

If an unsupported value is included in the `edb_audit_statement` parameter, then an error occurs when applying the setting to the database server. See the database server log file for the error such as in the following example in which `create materialized vw` results in the error. The correct value is `create materialized view`.

```
2017-07-16 11:20:39 EDT LOG:  invalid value for parameter
"edb_audit_statement": "create materialized vw, create sequence, grant"
2017-07-16 11:20:39 EDT FATAL:  configuration file "/var/lib/edb/as14/data/
postgresql.conf" contains errors
```

DDL and DCL statements

When auditing DDL and DCL statements, the following general rules apply to the values that can be included in the `edb_audit_statement` parameter:

- If the `edb_audit_statement` parameter includes `ddl` or `all`, then all DDL statements are audited. In addition, the DCL statements `GRANT` and `REVOKE` are audited.
- If the `edb_audit_statement` is set to `none`, then no DDL nor DCL statements are audited.
- You can choose specific types of DDL and DCL statements for auditing by including a combination of values in the `edb_audit_statement` parameter.

Use the following syntax to specify an `edb_audit_statement` parameter value for DDL statements:

```
{ create | alter | drop } [ <object_type> ]
```

`object_type` is any of the following:

- `ACCESS METHOD`
- `AGGREGATE`
- `CAST`
- `COLLATION`
- `CONVERSION`
- `DATABASE`
- `EVENT TRIGGER`
- `EXTENSION`
- `FOREIGN TABLE`
- `FUNCTION`
- `INDEX`
- `LANGUAGE`
- `LARGE OBJECT`
- `MATERIALIZED VIEW`
- `OPERATOR`
- `OPERATOR CLASS`

- OPERATOR FAMILY
- POLICY
- PUBLICATION
- ROLE
- RULE
- SCHEMA
- SEQUENCE
- SERVER
- SUBSCRIPTION
- TABLE
- TABLESPACE
- TEXT SEARCH CONFIGURATION
- TEXT SEARCH DICTIONARY
- TEXT SEARCH PARSER
- TEXT SEARCH TEMPLATE
- TRANSFORM
- TRIGGER
- TYPE
- USER MAPPING
- VIEW

Descriptions of object types as used in SQL commands can be found in the [PostgreSQL core documentation](#).

If `object_type` is omitted from the parameter value, then all of the specified command statements (`create`, `alter`, or `drop`) are audited.

Use the following syntax to specify an `edb_audit_statement` parameter value for DCL statements:

```
{ grant | revoke }
```

Examples

In this example, `edb_audit_connect` and `edb_audit_statement` are set with the following non-default values:

```
logging_collector = on
edb_audit_connect = all
edb_audit_statement = create, alter, error
```

Thus, only SQL statements invoked by the `CREATE` and `ALTER` commands are audited. Error messages are also included in the audit log.

The following is the database session that occurs:

```
$ psql edb enterprisedb
Password for user enterprisedb:
psql.bin (14.0.0)
Type "help" for help.
```

```
edb=# SHOW edb_audit_connect;
__OUTPUT__
edb_audit_connect
- -----
all
(1 row)
```

```
edb=# SHOW edb_audit_statement;
__OUTPUT__
edb_audit_statement
- -----
create, alter, error
(1 row)
```

```
edb=# CREATE ROLE adminuser;
CREATE ROLE
edb=# ALTER ROLE adminuser WITH LOGIN, SUPERUSER, PASSWORD password;
ERROR:  syntax error at or near ","
LINE 1: ALTER ROLE adminuser WITH LOGIN, SUPERUSER, PASSWORD passwo...
                                     ^
edb=# ALTER ROLE adminuser WITH LOGIN SUPERUSER PASSWORD password;
ALTER ROLE
edb=# CREATE DATABASE auditdb;
CREATE DATABASE
edb=# ALTER DATABASE auditdb OWNER TO adminuser;
ALTER DATABASE
edb=# \c auditdb adminuser
Password for user adminuser:
You are now connected to database "auditdb" as user "adminuser".
auditdb=# CREATE SCHEMA edb;
CREATE SCHEMA
auditdb=# SET search_path TO edb;
SET
auditdb=# CREATE TABLE department (
auditdb(# deptno          NUMBER(2) NOT NULL CONSTRAINT dept_pk PRIMARY KEY,
auditdb(#  dname          VARCHAR2(14) CONSTRAINT dept_dname_uq UNIQUE,
auditdb(#  loc            VARCHAR2(13)
auditdb(# );
CREATE TABLE
auditdb=# DROP TABLE department;
DROP TABLE
auditdb=# CREATE TABLE dept (
auditdb(# deptno          NUMBER(2) NOT NULL CONSTRAINT dept_pk PRIMARY KEY,
auditdb(#  dname          VARCHAR2(14) CONSTRAINT dept_dname_uq UNIQUE,
auditdb(#  loc            VARCHAR2(13)
auditdb(# );
CREATE TABLE
```

The resulting audit log file contains the following. (Each audit log entry was split and displays across multiple lines. A blank line was inserted between the audit log entries for visual clarity.)

```

2020-05-25 12:32:22.799 IST,"enterprisedb","edb",72518,"[local]",5ecb6d7e.
11b46,1,"authentication",2020-05-25 12:32:22 IST,4/19,0,AUDIT,000000,
"connection authorized:user=enterprisedb database=edb",,,,,,,,,,"client
backend",,,,,,"connect"

2020-05-25 12:34:05.843 IST,"enterprisedb","edb",72518,"[local]",5ecb6d7e.
11b46,2,"idle",2020-05-25 12:32:22 IST,4/23,0,AUDIT,000000,"statement: CREATE
ROLE adminuser;",,,,,,,,,,"psql","client backend","CREATE ROLE","create"

2020-05-25 12:34:16.617 IST,"enterprisedb","edb",72518,"[local]",5ecb6d7e.
11b46,3,"idle",2020-05-25 12:32:22 IST,4/24,0,ERROR,42601,"syntax error at or
near """,,,,,,"ALTER ROLE adminuser WITH LOGIN, SUPERUSER, PASSWORD
password;",32,,,"psql","client backend",,,,,,"error"

2020-05-25 12:34:29.954 IST,"enterprisedb","edb",72518,"[local]",5ecb6d7e.
11b46,4,"idle",2020-05-25 12:32:22 IST,4/25,0,AUDIT,000000,"statement: ALTER
ROLE adminuser WITH LOGIN SUPERUSER PASSWORD password;",,,,,,,,,,"psql",
"client backend","ALTER ROLE","alter"

2020-05-25 12:34:40.114 IST,"enterprisedb","edb",72518,"[local]",5ecb6d7e.
11b46,5,"idle",2020-05-25 12:32:22 IST,4/26,0,AUDIT,000000,"statement: CREATE
DATABASE auditdb;",,,,,,,,,,"psql","client backend","CREATE DATABASE","create"

2020-05-25 12:34:50.591 IST,"enterprisedb","edb",72518,"[local]",5ecb6d7e.
11b46,6,"idle",2020-05-25 12:32:22 IST,4/27,0,AUDIT,000000,"statement: ALTER
DATABASE auditdb OWNER TO adminuser;",,,,,,,,,,"psql","client backend","ALTER
DATABASE","alter"

2020-05-25 12:35:01.554 IST,"adminuser","auditdb",75531,"[local]",5ecb6e1d.
1270b,1,"authentication",2020-05-25 12:35:01 IST,5/11,0,AUDIT,000000,
"connection authorized:user=adminuser database=auditdb",,,,,,,,,,"client
backend",,,,,,"connect"

2020-05-25 12:35:12.931 IST,"adminuser","auditdb",75531,"[local]",5ecb6e1d.
1270b,2,"idle",2020-05-25 12:35:01 IST,5/13,0,AUDIT,000000,"statement: CREATE
SCHEMA edb;",,,,,,,,,,"psql","client backend","CREATE SCHEMA","create"

2020-05-25 12:37:18.547 IST,"adminuser","auditdb",75531,"[local]",5ecb6e1d.
1270b,3,"idle",2020-05-25 12:35:01 IST,5/15,0,AUDIT,000000,"statement: CREATE
TABLE department
(
    deptno    NUMBER(2) NOT NULL CONSTRAINT dept_pk PRIMARY KEY,
    dname     VARCHAR2(14) CONSTRAINT dept_dname_uq UNIQUE,
    loc       VARCHAR2(13)
);",,,,,,,,,,"psql","client backend","CREATE TABLE","create"

2020-05-25 12:39:09.065 IST,"adminuser","auditdb",75531,"[local]",5ecb6e1d.
1270b,4,"idle",2020-05-25 12:35:01 IST,5/17,0,AUDIT,000000,"statement: CREATE
TABLE dept (
    deptno    NUMBER(2) NOT NULL CONSTRAINT dept_pk PRIMARY KEY,
```

(continues on next page)

(continued from previous page)

```

        dname          VARCHAR2(14) CONSTRAINT dept_dname_uq UNIQUE,
        loc            VARCHAR2(13)
);,,,,,,,,,"psql","client backend","CREATE TABLE","create"

```

The CREATE and ALTER statements for the adminuser role and auditdb database are audited. Because error is included in the edb_audit_statement parameter, the error for the ALTER ROLE adminuser statement is also logged.

Similarly, the CREATE statements for schema edb and tables department and dept are audited.

There's no edb_audit_statement setting that results in auditing successfully processed DROP statements such as ddl, all, or drop. Thus the DROP TABLE department statement isn't in the audit log.

In this example, edb_audit_connect and edb_audit_statement are set with the following non-default values:

```

edb_audit_connect = on
edb_audit_connect = all
edb_audit_statement = create view,create materialized view,create sequence,grant

```

Thus, only SQL statements invoked by the CREATE VIEW, CREATE MATERIALIZED VIEW, CREATE SEQUENCE and GRANT commands are audited.

The following is the database session that occurs:

```

$ psql auditdb adminuser
Password for user adminuser:
psql.bin (14.0.0)
Type "help" for help.

auditdb=# SHOW edb_audit_connect;
__OUTPUT__
edb_audit_connect
-----
all
(1 row)

```

```

auditdb=# SHOW edb_audit_statement;
__OUTPUT__
          edb_audit_statement
-----
create view,create materialized view,create sequence,grant
(1 row)

```

```

auditdb=# SET search_path TO edb;
SET
auditdb=# CREATE TABLE emp (
auditdb(#   empno          NUMBER(4) NOT NULL CONSTRAINT emp_pk PRIMARY KEY,
auditdb(#   ename          VARCHAR2(10),
auditdb(#   job            VARCHAR2(9),
auditdb(#   mgr            NUMBER(4),
auditdb(#   hiredate       DATE,
auditdb(#   sal            NUMBER(7,2) CONSTRAINT emp_sal_ck CHECK (sal > 0),
auditdb(#   comm           NUMBER(7,2),
auditdb(#   deptno         NUMBER(2) CONSTRAINT emp_ref_dept_fk
auditdb(#                   REFERENCES dept(deptno)

```

(continues on next page)

(continued from previous page)

```

auditdb(# );
CREATE TABLE
auditdb=# CREATE VIEW salesemp AS
auditdb=# SELECT empno, ename, hiredate, sal, comm FROM emp WHERE job =
SALESMAN;
CREATE VIEW
auditdb=# CREATE MATERIALIZED VIEW managers AS
auditdb=# SELECT empno, ename, hiredate, sal, comm FROM emp WHERE job =
MANAGER;
SELECT 0
auditdb=# CREATE SEQUENCE next_empno START WITH 8000 INCREMENT BY 1;
CREATE SEQUENCE
auditdb=# GRANT ALL ON dept TO PUBLIC;
GRANT
auditdb=# GRANT ALL ON emp TO PUBLIC;
GRANT

```

The resulting audit log file contains the following. (Each audit log entry was split and displays across multiple lines. A blank line was inserted between the audit log entries for visual clarity.)

```

2020-05-25 14:05:29.163 IST,"adminuser","auditdb",40810,"[local]",5ecb8351.
9f6a,1,"authentication",2020-05-25 14:05:29 IST,4/28,0,AUDIT,000000,
"connection authorized:user=adminuser database=auditdb",,,,,,,,,,"client
backend",,,,,,"connect"

2020-05-25 14:12:06.318 IST,"adminuser","auditdb",40810,"[local]",5ecb8351.
9f6a,2,"idle",2020-05-25 14:05:29 IST,4/34,0,AUDIT,000000,"statement: CREATE
VIEW salesemp AS SELECT empno, ename, hiredate, sal, comm FROM emp WHERE job
= SALESMAN;",,,,,,,,,,"psql","client backend","CREATE VIEW","create"

2020-05-25 14:13:26.657 IST,"adminuser","auditdb",40810,"[local]",5ecb8351.
9f6a,3,"idle",2020-05-25 14:05:29 IST,4/36,0,AUDIT,000000,"statement: CREATE
MATERIALIZED VIEW managers AS SELECT empno, ename, hiredate, sal, comm FROM
emp WHERE job = MANAGER;",,,,,,,,,,"psql","client backend","CREATE
MATERIALIZED VIEW","create"

2020-05-25 14:13:38.928 IST,"adminuser","auditdb",40810,"[local]",5ecb8351.
9f6a,4,"idle",2020-05-25 14:05:29 IST,4/37,0,AUDIT,000000,"statement: CREATE
SEQUENCE next_empno START WITH 8000 INCREMENT BY 1;",,,,,,,,,,"psql","client
backend","CREATE SEQUENCE","create"

2020-05-25 14:13:51.434 IST,"adminuser","auditdb",40810,"[local]",5ecb8351.
9f6a,5,"idle",2020-05-25 14:05:29 IST,4/38,0,AUDIT,000000,"statement: GRANT
ALL ON dept TO PUBLIC;",,,,,,,,,,"psql","client backend","GRANT","grant"

2020-05-25 14:14:03.737 IST,"adminuser","auditdb",40810,"[local]",5ecb8351.
9f6a,6,"idle",2020-05-25 14:05:29 IST,4/39,0,AUDIT,000000,"statement: GRANT
ALL ON emp TO PUBLIC;",,,,,,,,,,"psql","client backend","GRANT","grant"

```

The CREATE VIEW and CREATE MATERIALIZED VIEW statements are audited. The prior CREATE TABLE emp statement isn't audited because none of the values create , create table , ddl , nor all are included in the edb_audit_statement parameter.

The `CREATE SEQUENCE` and `GRANT` statements are audited because those values are included in the `edb_audit_statement` parameter.

In this example, `edb_audit_connect` and `edb_audit_statement` are set with the following non-default values:

```
logging_collector = on
edb_audit_connect = all
edb_audit_statement = none
```

The session for users who connect as `ADMIN` or `SYSDBA` can be fully audited. An admin user is connected to a database `auditdb` as `ADMINUSER`. The following `ALTER USER` command specifies to audit `ADMINUSER`.

```
ALTER USER adminuser SET edb_audit_statement = "all";
```

Setting the `edb_audit_statement` parameter to `all` allows auditing of all of the SQL statements for an admin user.

The following is the database session that occurs:

```
$ psql auditdb adminuser
Password for user adminuser:
psql.bin (14.0.0)
Type "help" for help.

auditdb=# SHOW edb_audit_connect;
__OUTPUT__
edb_audit_connect
- -----
all
(1 row)
```

```
auditdb=# SHOW edb_audit_statement;
__OUTPUT__
edb_audit_statement
- -----
all
(1 row)
```

```
auditdb=# SET search_path TO edb;
SET
auditdb=> CREATE TABLE dept
auditdb->      (deptno NUMBER(2),
auditdb(>      dname VARCHAR2(14),
auditdb(>      loc VARCHAR2(13) );
CREATE TABLE
auditdb=> INSERT INTO dept VALUES (10, ACCOUNTING, NEW YORK);
INSERT 0 1
auditdb=> INSERT INTO dept VALUES (20, RESEARCH, DALLAS);
INSERT 0 1
auditdb=> INSERT INTO dept VALUES (30, SALES, CHICAGO);
INSERT 0 1
auditdb=> UPDATE dept SET loc = BEDFORD WHERE deptno = 20;
UPDATE 1
auditdb=> SELECT * FROM dept;
__OUTPUT__
```

(continues on next page)

(continued from previous page)

deptno	dname	loc
10	ACCOUNTING	NEW YORK
30	SALES	CHICAGO
20	RESEARCH	BEDFORD

(3 rows)

```
auditdb=> DELETE FROM emp WHERE deptno = 40;
ERROR: relation "emp" does not exist
LINE 1: DELETE FROM emp WHERE deptno = 40;
              ^

auditdb=> DELETE FROM dept WHERE deptno = 10;
DELETE 1
auditdb=> SELECT * FROM dept;
__OUTPUT__
deptno | dname   | loc
-----+-----+-----
    30 | SALES   | CHICAGO
    20 | RESEARCH | BEDFORD
(2 rows)
```

The resulting audit log file contains the following. (Each audit log entry was split and displays across multiple lines. A blank line was inserted between the audit log entries for visual clarity.)

```
2021-06-23 06:06:59.027 IST,"adminuser","auditdb",60218,"[local]",60d3083b.
eb3a,1,"authentication",2021-06-23 06:06:59 IST,4/19,0,AUDIT,000000,"connection_
↪authorized:
user=adminuser database=auditdb",,,,,,,,,,"client backend",,,"","connect"

2021-06-23 06:07:33.192 IST,"adminuser","auditdb",66316,"[local]",60daab0c.
1030c,2,"idle",2021-06-23 06:07:33 IST,4/16,0,AUDIT,000000,"statement: SHOW edb_audit_
↪connect;
",,,,,,,,,,"psql","client backend",,"SHOW",,,"","sql statement"

2021-06-23 06:08:12.474 IST,"adminuser","auditdb",66316,"[local]",60daab0c.
1030c,3,"idle",2021-06-23 06:08:12 IST,4/17,0,AUDIT,000000,"statement: SHOW edb_audit_
↪statement;
",,,,,,,,,,"psql","client backend",,"SHOW",,,"","sql statement"

2021-06-23 06:08:20.519 IST,"adminuser","auditdb",66922,"[local]",60dab036.
1056a,4,"idle",2021-06-23 06:08:20 IST,4/15,0,AUDIT,000000,"statement: SET search_path TO_
↪edb;
",,,,,,,,,,"psql","client backend",,"SET",,,"","set"

2021-06-23 06:09:27.613 IST,"adminuser","auditdb",60218,"[local]",60dab117.
10602,5,"idle",2021-06-23 06:09:59 IST,4/21,0,AUDIT,000000,"statement: CREATE TABLE dept
(deptno NUMBER(2),
dname VARCHAR2(14),
loc VARCHAR2(13) );",,,,,,,,,,"psql","client backend",,"CREATE TABLE",,,"","create"

2021-06-23 06:09:39.238 IST,"adminuser","auditdb",60218,"[local]",60d3083b.
eb3a,6,"idle",2021-06-23 06:09:29 IST,4/22,0,AUDIT,000000,"statement: INSERT INTO
```

(continues on next page)

(continued from previous page)

```

dept VALUES (10, ACCOUNTING, NEW YORK);",,,,,,,,,,"psql","client backend",,"INSERT","",
↪ "insert"

2021-06-23 06:09:39.242 IST,"adminuser","auditdb",60218,"[local]",60d3083b.
eb3a,7,"idle",2021-06-23 06:09:29 IST,4/23,0,AUDIT,000000,"statement: INSERT INTO
dept VALUES (20, RESEARCH, DALLAS);",,,,,,,,,,"psql","client backend",,"INSERT","", "insert
↪ "

2021-06-23 06:09:39.247 IST,"adminuser","auditdb",60218,"[local]",60d3083b.
eb3a,8,"idle",2021-06-23 06:08:35 IST,4/24,0,AUDIT,000000,"statement: INSERT INTO
dept VALUES (30, SALES, CHICAGO);",,,,,,,,,,"psql","client backend",,"INSERT","", "insert"

2021-06-23 06:10:04.849 IST,"adminuser","auditdb",60218,"[local]",60d3083b.
eb3a,9,"idle",2021-06-23 06:08:59 IST,4/25,0,AUDIT,000000,"statement: UPDATE dept SET loc
↪ = BEDFORD
WHERE deptno = 20;",,,,,,,,,,"psql","client backend",,"UPDATE","", "update"

2021-06-23 06:10:16.045 IST,"adminuser","auditdb",60218,"[local]",60d3083b.
eb3a,10,"idle",2021-06-23 06:08:59 IST,4/26,0,AUDIT,000000,"statement: SELECT * FROM dept;
↪ ",,,,,,,,,,
"psql","client backend",,"SELECT","", "select"

2021-06-23 06:10:40.593 IST,"adminuser","auditdb",60218,"[local]",60d3083b.
eb3a,11,"idle",2021-06-23 06:08:59 IST,4/27,0,AUDIT,000000,"statement: DELETE FROM emp
↪ WHERE deptno = 40;
",,,,,,,,,,"psql","client backend",,"DELETE","", "delete"

2021-06-23 06:10:40.594 IST,"adminuser","auditdb",60218,"[local]",60d3083b.
eb3a,12,"DELETE",2021-06-23 06:08:59 IST,4/27,0,ERROR,42P01,"relation ""emp"" does not
↪ exist",,,,,,
"DELETE FROM emp WHERE deptno = 40;",13,,,"psql","client backend",,"DELETE","", "error"

2021-06-23 06:11:02.563 IST,"adminuser","auditdb",60218,"[local]",60d3083b.
eb3a,13,"idle",2021-06-23 06:08:59 IST,4/28,0,AUDIT,000000,"statement: DELETE FROM dept
↪ WHERE deptno = 10;
",,,,,,,,,,"psql","client backend",,"DELETE","", "delete"

2021-06-23 06:11:14.585 IST,"adminuser","auditdb",60218,"[local]",60d3083b.
eb3a,14,"idle",2021-06-23 06:08:59 IST,4/29,0,AUDIT,000000,"statement: SELECT * FROM dept;
↪ ",,,,,,,,,,
"psql","client backend",,"SELECT","", "select"

```

In this example, `edb_audit_connect` and `edb_audit_statement` are set with the following non-default values:

```

logging_collector = on
edb_audit_connect = all
edb_audit_statement = all

```

The audit session for a user `carol` can be fully blocked by the database administrators using the `ALTER USER` command:

```

ALTER USER carol SET edb_audit_statement = "none";

```

Note: The database administrator can allow a specific user to audit any SQL statements by specifying the *ALTER USER* command and setting the *edb_audit_statement* parameter to any value. The following is the database session that occurs:

```
$ psql auditdb carol
Password for user carol:
psql.bin (14.0.0)
Type "help" for help.

auditdb=# SHOW edb_audit_connect;
__OUTPUT__
edb_audit_connect
- -----
all
(1 row)
```

```
auditdb=# SHOW edb_audit_statement;
__OUTPUT__
edb_audit_statement
- -----
none
(1 row)
```

```
auditdb=# SET search_path TO edb;
SET
auditdb=> CREATE TABLE salgrade
auditdb->      (grade NUMBER,
auditdb(>      losal NUMBER,
auditdb(>      hisal NUMBER);
CREATE TABLE
INSERT INTO salgrade VALUES (1, 700, 1200);
INSERT INTO salgrade VALUES (2, 1201, 1400);
INSERT INTO salgrade VALUES (3, 1401, 2000);
```

The resulting audit log file contains only the connection authentication information. Setting the *edb_audit_statement* parameter to *none* doesn't allow the auditing of SQL statements for *carol*, so no audit logs are generated.

(Each audit log entry was split and displays across multiple lines. A blank line was inserted between the audit log entries for visual clarity.)

```
2021-06-29 02:27:26.240 IST,"carol","auditdb",68072,"[local]",60dabd4e.
109e8,1,"authentication",2021-06-29 02:27:26 IST,4/13,0,AUDIT,00000,"connection_
→authorized: user=carol
database=auditdb",,,,,,,,,,"client backend",,,,,,"connect"
```

DML statements

When auditing DML statements, the following general rules apply to the values that can be included in the `edb_audit_statement` parameter:

- If the `edb_audit_statement` parameter includes `dml` or `all`, then all DML statements are audited.
- If the `edb_audit_statement` is set to `none`, then no DML statements are audited.
- You can choose specific types of DML statements for auditing by including a combination of values in the `edb_audit_statement` parameter.

Use the following syntax to specify an `edb_audit_statement` parameter value for SQL `INSERT`, `UPDATE`, `DELETE`, or `TRUNCATE` statements:

```
{ insert | update | delete | truncate }
```

Example

In this example, `edb_audit_connect` and `edb_audit_statement` are set with the following non-default values:

```
logging_collector = on
edb_audit_connect = all
edb_audit_statement = UPDATE, DELETE, error
```

Thus, only SQL statements invoked by the `UPDATE` and `DELETE` commands are audited. All errors are also included in the audit log, including errors not related to the `UPDATE` and `DELETE` commands.

The following is the database session that occurs:

```
$ psql auditdb adminuser
Password for user adminuser:
psql.bin (14.0.0)
Type "help" for help.

auditdb=# SHOW edb_audit_connect;
__OUTPUT__
edb_audit_connect
- -----
all
(1 row)
```

```
auditdb=# SHOW edb_audit_statement;
__OUTPUT__
edb_audit_statement
- -----
UPDATE, DELETE, error
(1 row)
```

```
auditdb=# SET search_path TO edb;
SET
auditdb=# INSERT INTO dept VALUES (10,ACCOUNTING,NEW YORK);
INSERT 0 1
auditdb=# INSERT INTO dept VALUES (20,RESEARCH,DALLAS);
```

(continues on next page)

(continued from previous page)

```

INSERT 0 1
auditdb=# INSERT INTO dept VALUES (30,SALES,CHICAGO);
INSERT 0 1
auditdb=# INSERT INTO dept VALUES (40,OPERATIONS,BOSTON);
INSERT 0 1
auditdb=# INSERT INTO emp VALUES (7369,SMITH,CLERK,7902,17-DEC-80,800,NULL,20);
INSERT 0 1
auditdb=# INSERT INTO emp VALUES (7499,ALLEN,SALESMAN,7698,20-FEB-81,1600,300,30);
INSERT 0 1
auditdb=# INSERT INTO emp VALUES (7521,WARD,SALESMAN,7698,22-FEB-81,1250,500,30);
INSERT 0 1
      .
      .
      .
auditdb=# INSERT INTO emp VALUES (7934,MILLER,CLERK,7782,23-JAN-82,1300,NULL,10);
INSERT 0 1
auditdb=# UPDATE dept SET loc = BEDFORD WHERE deptno = 40;
UPDATE 1
auditdb=# SELECT * FROM dept;
__OUTPUT__
 deptno |   dname   |   loc
-----+-----+-----
    10 | ACCOUNTING | NEW YORK
    20 | RESEARCH  | DALLAS
    30 | SALES     | CHICAGO
    40 | OPERATIONS | BEDFORD
(4 rows)

```

```

auditdb=# DELETE FROM emp WHERE deptno = 10;
DELETE 3
auditdb=# TRUNCATE employee;
ERROR:  relation "employee" does not exist
auditdb=# TRUNCATE emp;
TRUNCATE TABLE
auditdb=# \q

```

The resulting audit log file contains the following. (Each audit log entry was split and displays across multiple lines. A blank line was inserted between the audit log entries for visual clarity.)

```

2020-05-25 15:17:16.392 IST,"adminuser","auditdb",123420,"[local]",5ecb9424.
1e21c,1,"authentication",2020-05-25 15:17:16 IST,5/3,0,AUDIT,000000,
"connection authorized:user=adminuser database=auditdb",,,,,,,,,,"client
backend",,,,,,connect"

2020-05-25 15:19:18.066 IST,"adminuser","auditdb",123420,"[local]",5ecb9424.
1e21c,2,"idle",2020-05-25 15:17:16 IST,5/14,0,AUDIT,000000,"statement: UPDATE
dept SET loc = BEDFORD WHERE deptno = 40;",,,,,,,,,,"psql","client backend",
"UPDATE","update"

2020-05-25 15:19:38.524 IST,"adminuser","auditdb",123420,"[local]",5ecb9424.
1e21c,3,"idle",2020-05-25 15:17:16 IST,5/16,0,AUDIT,000000,"statement: DELETE
FROM emp WHERE deptno = 10;",,,,,,,,,,"psql","client backend","DELETE","delete"

```

(continues on next page)

(continued from previous page)

```
2020-05-25 15:19:48.149 IST,"adminuser","auditdb",123420,"[local]",5ecb9424.
1e21c,4,"TRUNCATE TABLE",2020-05-25 15:17:16 IST,5/17,0,ERROR,42P01,"relation
""employee""does not exist",,,,,,"TRUNCATE employee;",,,,"psql","client backend
","TRUNCATE TABLE","error"
```

The UPDATE dept and DELETE FROM emp statements are audited. All of the prior INSERT statements aren't audited because the values insert , dml , or all aren't included in the edb_audit_statement parameter.

The SELECT * FROM dept statement isn't audited because select and all aren't included in the edb_audit_statement parameter.

Because error is specified in the edb_audit_statement parameter but not the truncate value, the error on the TRUNCATE employee statement is logged in the audit file. But the successful TRUNCATE emp statement isn't.

3.5.3 Enabling audit logging

You can configure EDB Postgres Advanced Server to log all connections, disconnections, DDL statements, DCL statements, DML statements, and any statements resulting in an error.

- Enable auditing by setting the edb_audit parameter to xml or csv .
- Set the file rotation day when the new file is created by setting the parameter edb_audit_rotation_day to the desired value.
- To audit all connections, set the parameter edb_audit_connect to all .
- To audit all disconnections, set the parameter edb_audit_disconnect to all .
- To audit DDL, DCL, DML and other statements, set the parameter edb_audit_statement according to the instructions in

Selecting SQL statements to audit .

Setting the edb_audit_statement parameter in the configuration file affects the entire database cluster.

You can further refine the type of statements that are audited. The type is controlled by the edb_audit_statement parameter according to the database in use as well as the database role running the session:

- You can set the edb_audit_statement parameter as an attribute of a specified database with the ALTER DATABASE dbname SET edb_audit_statement command. This setting overrides the edb_audit_statement parameter in the configuration file for statements executed when connected to database dbname .
- You can set the edb_audit_statement parameter as an attribute of a specified role with the ALTER ROLE rolename SET edb_audit_statement command. This setting overrides the edb_audit_statement parameter in the configuration file as well as any setting assigned to the database by the ALTER DATABASE command when the specified role is running the current session.
- You can set the edb_audit_statement parameter as an attribute of a specified role when using a specified database with the ALTER ROLE rolename IN DATABASE dbname SET edb_audit_statement command. This setting overrides the edb_audit_statement parameter in the configuration file. It also overrides any setting assigned to the database by the ALTER DATABASE command and as any setting assigned to the role with the ALTER ROLE command without the IN DATABASE clause.

Examples

The following are examples of this technique.

The database cluster is established with `edb_audit_statement` set to `all` as shown in its `postgresql.conf` file:

```
logging_collector = on
edb_audit_statement = all      # Statement type to be audited:
                               # none, dml, insert, update, delete, truncate,
                               # select, error, rollback, ddl, create, drop,
                               # alter, grant, revoke, set, all
                               # {select | update | delete | insert}@groupname
```

A database and role are established with the following settings for the `edb_audit_statement` parameter:

- Database `auditdb` with `ddl`, `insert`, `update`, and `delete`
- Role `admin` with `select`, `truncate`, and `set`
- Role `admin` in database `auditdb` with `create table`, `insert`, and `update`

The following shows creating and altering the database and role:

```
$ psql edb enterprisedb
Password for user enterprisedb:
psql.bin (14.0.0)
Type "help" for help.

edb=# SHOW edb_audit_statement;
__OUTPUT__
edb_audit_statement
- -----
all
(1 row)
```

```
edb=# CREATE DATABASE auditdb;
CREATE DATABASE
edb=# ALTER DATABASE auditdb SET edb_audit_statement TO ddl, insert, update, delete;
ALTER DATABASE
edb=# CREATE ROLE admin WITH LOGIN SUPERUSER PASSWORD password;
CREATE ROLE
edb=# ALTER ROLE admin SET edb_audit_statement TO select, truncate;
ALTER ROLE
edb=# ALTER ROLE admin IN DATABASE auditdb SET edb_audit_statement TO create table,
↪insert, update;
ALTER ROLE
```

The following shows the changes made and the resulting audit log file for three cases.

Case 1: Changes made in database auditdb by role enterprisedb

Only ddl , insert , update , and delete statements are audited:

```
$ psql auditdb enterprisedb
Password for user enterprisedb:
psql.bin (14.0.0)
Type "help" for help.

auditdb=# SHOW edb_audit_statement;
__OUTPUT__
      edb_audit_statement
-----
ddl, insert, update, delete
(1 row)
```

```
auditdb=# CREATE TABLE audit_tbl (f1 INTEGER PRIMARY KEY, f2 TEXT);
CREATE TABLE
auditdb=# INSERT INTO audit_tbl VALUES (1, Row 1);
INSERT 0 1
auditdb=# UPDATE audit_tbl SET f2 = Row A WHERE f1 = 1;
UPDATE 1
auditdb=# SELECT * FROM audit_tbl;
__OUTPUT__
 f1 | f2
-----+-----
  1 | Row A
(1 row)

<== Should not be audited
```

```
auditdb=# TRUNCATE audit_tbl;
TRUNCATE TABLE

<== Should not be audited
```

The following audit log file shows entries only for the CREATE TABLE , INSERT INTO audit_tbl , and UPDATE audit_tbl statements. The SELECT * FROM audit_tbl and TRUNCATE audit_tbl statements weren't audited. (Each audit log entry was split and displays across multiple lines. A blank line was inserted between the audit log entries for visual clarity.)

```
2020-05-25 15:59:12.332 IST,"enterprisedb","auditdb",41837,"[local]",5ecb9dee.a36d,2,
"idle",2020-05-25 15:59:02 IST,5/7,0,AUDIT,00000,"statement: CREATE TABLE
audit_tbl(f1 INTEGER PRIMARY KEY, f2 TEXT);",,,,,,,,,,"psql","client backend","CREATE_
↪TABLE","", "create"

2020-05-25 15:59:22.419 IST,"enterprisedb","auditdb",41837,"[local]",5ecb9dee.a36d,3,
"idle",2020-05-25 15:59:02 IST,5/8,0,AUDIT,00000,"statement: INSERT INTO
audit_tbl VALUES (1, Row 1);",,,,,,,,,,"psql","client backend","INSERT","", "insert"

2020-05-25 15:59:32.180 IST,"enterprisedb","auditdb",41837,"[local]",5ecb9dee.a36d,4,
"idle",2020-05-25 15:59:02 IST,5/9,0,AUDIT,00000,"statement: UPDATE
audit_tbl SET f2 = Row A WHERE f1 = 1;",,,,,,,,,,"psql","client backend","UPDATE","",
↪"update"
```

Case 2: Changes made in database edb by role admin

Only select , truncate , and set statements are audited:

```
$ psql edb admin
Password for user admin:
psql.bin (14.0.0)
Type "help" for help.

edb=# SHOW edb_audit_statement;
__OUTPUT__
edb_audit_statement
- - - - -
select, truncate, set
(1 row)
```

```
edb=# SET default_with_rowids TO TRUE;
SET

edb=# CREATE TABLE edb_tbl (f1 INTEGER PRIMARY KEY, f2 TEXT); <== Should not be audited
CREATE TABLE
edb=# INSERT INTO edb_tbl VALUES (1, Row 1);                                <== Should not be audited
INSERT 0 1
edb=# SELECT * FROM edb_tbl;
__OUTPUT__
 f1 | f2
- -+-----
  1 | Row 1
(1 row)
```

```
edb=# TRUNCATE edb_tbl;
TRUNCATE TABLE
```

Continuation of the audit log file now appears as follows. The last two entries representing the second case show only the SET default_with_rowids TO TRUE, SELECT * FROM edb_tbl, and TRUNCATE edb_tbl statements. The CREATE TABLE edb_tbl and INSERT INTO edb_tbl statements weren't audited.

```
2020-05-25 15:59:12.332 IST,"enterprisedb","auditdb",41837,"[local]",5ecb9dee.a36d,2,
"idle",2020-05-25 15:59:02 IST,5/7,0,AUDIT,00000,"statement: CREATE TABLE
audit_tbl(f1 INTEGER PRIMARY KEY, f2 TEXT);",,,,,,,,,,"psql","client backend","CREATE_
↪TABLE","", "create"

2020-05-25 15:59:22.419 IST,"enterprisedb","auditdb",41837,"[local]",5ecb9dee.a36d,3,
"idle",2020-05-25 15:59:02 IST,5/8,0,AUDIT,00000,"statement: INSERT INTO
audit_tbl VALUES (1, Row 1);",,,,,,,,,,"psql","client backend","INSERT","", "insert"

2020-05-25 15:59:32.180 IST,"enterprisedb","auditdb",41837,"[local]",5ecb9dee.a36d,4,
"idle",2020-05-25 15:59:02 IST,5/9,0,AUDIT,00000,"statement: UPDATE
audit_tbl SET f2 = Row A WHERE f1 = 1;",,,,,,,,,,"psql","client backend","UPDATE","",
↪"update"

2021-02-18 08:04:57.434 IST,"admin","edb",3182,"[local]",602e65dc.c6e,1,
"idle",2021-02-18 08:04:28 IST,4/22,0,AUDIT,00000,"statement: SET default_with_rowids TO_
↪TRUE;
```

(continues on next page)

(continued from previous page)

```

",,,,,,,,,,"psql","client backend","SET","", "set"

2021-02-18 08:06:01.662 IST,"admin","edb",3182,"[local]",602e65dc.c6e,2,
"idle",2021-02-18 08:04:28 IST,4/27,0,AUDIT,00000,"statement: SELECT * FROM edb_tbl;
",,,,,,,,,,"psql","client backend","SELECT","", "select"

2021-02-18 08:06:11.125 IST,"admin","edb",3182,"[local]",602e65dc.c6e,3,
"idle",2021-02-18 08:04:28 IST,4/28,0,AUDIT,00000,"statement: TRUNCATE edb_tbl;
",,,,,,,,,,"psql","client backend","TRUNCATE TABLE","", "truncate"

```

Case 3: Changes made in database auditdb by role admin

Only create table, insert, and update statements are audited:

```

$ psql auditdb admin
Password for user admin:
psql.bin (14.0.0)
Type "help" for help.

auditdb=# SHOW edb_audit_statement;
__OUTPUT__
      edb_audit_statement
-----
create table, insert, update
(1 row)

```

```

auditdb=# CREATE TABLE audit_tbl_2 (f1 INTEGER PRIMARY KEY, f2 TEXT);
CREATE TABLE
auditdb=# INSERT INTO audit_tbl_2 VALUES (1, Row 1);
INSERT 0 1
auditdb=# SELECT * FROM audit_tbl_2;                <== Should not be audited
__OUTPUT__
 f1 | f2
- -+-----
  1 | Row 1
(1 row)

```

```

auditdb=# TRUNCATE audit_tbl_2;                      <== Should not be audited
TRUNCATE TABLE

```

Continuation of the audit log file now appears as follows. The next-to-last two entries representing the third case show only CREATE TABLE audit_tbl_2 and INSERT INTO audit_tbl_2 statements. The SELECT * FROM audit_tbl_2 and TRUNCATE audit_tbl_2 statements weren't audited.

```

2020-05-25 15:59:12.332 IST,"enterprisedb","auditdb",41837,"[local]",5ecb9dee.a36d,2,
"idle",2020-05-25 15:59:02 IST,5/7,0,AUDIT,00000,"statement: CREATE TABLE
audit_tbl(f1 INTEGER PRIMARY KEY, f2 TEXT);",,,,,,,,,,"psql","client backend","CREATE_
↪TABLE","", "create"

2020-05-25 15:59:22.419 IST,"enterprisedb","auditdb",41837,"[local]",5ecb9dee.a36d,3,

```

(continues on next page)

(continued from previous page)

```

"idle",2020-05-25 15:59:02 IST,5/8,0,AUDIT,00000,"statement: INSERT INTO
audit_tbl VALUES (1, Row 1);",,,,,,,,,,"psql","client backend","INSERT","", "insert"

2020-05-25 15:59:32.180 IST,"enterprisedb","auditdb",41837,"[local]",5ecb9dee.a36d,4,
"idle",2020-05-25 15:59:02 IST,5/9,0,AUDIT,00000,"statement: UPDATE
audit_tbl SET f2 = Row A WHERE f1 = 1;",,,,,,,,,,"psql","client backend","UPDATE","",
↪ "update"

2021-02-18 08:04:57.434 IST,"admin","edb",3182,"[local]",602e65dc.c6e,1,
"idle",2021-02-18 08:04:28 IST,4/22,0,AUDIT,00000,"statement: SET default_with_rowids TO_
↪ TRUE;
",,,,,,,,,,"psql","client backend","SET","", "set"

2021-02-18 08:06:01.662 IST,"admin","edb",3182,"[local]",602e65dc.c6e,2,
"idle",2021-02-18 08:04:28 IST,4/27,0,AUDIT,00000,"statement: SELECT * FROM edb_tbl;
",,,,,,,,,,"psql","client backend","SELECT","", "select"

2021-02-18 08:06:11.125 IST,"admin","edb",3182,"[local]",602e65dc.c6e,3,
"idle",2021-02-18 08:04:28 IST,4/28,0,AUDIT,00000,"statement: TRUNCATE edb_tbl;
",,,,,,,,,,"psql","client backend","TRUNCATE TABLE","", "truncate"

2020-05-25 17:30:59.057 IST,"admin","auditdb",122093,"[local]",5ecbb370.1dced,2,
"idle",2020-05-25 17:30:48 IST,5/11,0,AUDIT,00000,"statement: CREATE TABLE
audit_tbl_2 (f1 INTEGER PRIMARY KEY, f2 TEXT);",,,,,,,,,,"psql","client backend","CREATE_
↪ TABLE","", "create"

2020-05-25 17:31:08.866 IST,"admin","auditdb",122093,"[local]",5ecbb370.1dced,3,
"idle",2020-05-25 17:30:48 IST,5/12,0,AUDIT,00000,"statement: INSERT INTO
audit_tbl_2 VALUES (1, Row 1);",,,,,,,,,,"psql","client backend","INSERT","", "insert"

```

3.5.4 Audit log file

You can generate the audit log file in CSV or XML format. The format is determined by the `edb_audit` configuration parameter.

The information in the audit log is based on the logging performed by PostgreSQL, as described in “Using CSV-Format Log Output” under “Error Reporting and Logging” in the [PostgreSQL core documentation](#).

The following table lists the fields in the order they appear in the CSV audit log format. The table contains the following information:

- Field. Name of the field as shown in the sample table definition in the PostgreSQL documentation.
- XML element/attribute. For the XML format, name of the XML element and its attribute (if used), referencing the value.
- Data type. Data type of the field as given by the PostgreSQL sample table definition.
- Description. Description of the field.

The fields that don’t have any values for logging appear as consecutive commas (,) in the CSV format.

Field	XML element/attribute	Data type	Description
<i>log_time</i>	<i>event/log_time</i>	<i>timestamp with time zone</i>	Log date/time of the statement.
<i>user_name</i>	<i>event/user</i>	<i>text</i>	Database user who executed the statement.
<i>database_name</i>	<i>event/database</i>	<i>text</i>	Database in which the statement was executed.
<i>process_id</i>	<i>event/process_id</i>	<i>integer</i>	Operating system process ID in which the statement was executed.
<i>connection_from</i>	<i>event/remote_host</i>	<i>text</i>	Host and port location from where the statement was executed.
<i>session_id</i>	<i>event/session_id</i>	<i>text</i>	Session ID in which the statement was executed.
<i>session_line_num</i>	<i>event/session_line_num</i>	<i>bigint</i>	Order of the statement within the session.
<i>process_status</i>	<i>event/process_status</i>	<i>text</i>	Processing status.
<i>session_start_time</i>	<i>event/session_start_time</i>	<i>timestamp with time zone</i>	Date/time when the session was started.
<i>virtual_transaction_id</i>	<i>event/virtual_transaction_id</i>	<i>text</i>	Virtual transaction ID of the statement.
<i>transaction_id</i>	<i>event/transaction_id</i>	<i>bigint</i>	Regular transaction ID of the statement.
<i>error_severity</i>	<i>error_severity</i>	<i>text</i>	Statement severity. Values are <i>AUDIT</i> for audited statements and <i>ERROR</i> for any resulting error messages.
<i>sql_state_code</i>	<i>event/sql_state_code</i>	<i>text</i>	SQL state code returned for the statement. The <i>sql_state_code</i> isn't logged when its value is 00000 for XML log format.
<i>message</i>	<i>message</i>	<i>text</i>	The SQL statement that was attempted for execution.
<i>detail</i>	<i>detail</i>	<i>text</i>	Error message detail.
<i>hint</i>	<i>hint</i>	<i>text</i>	Hint for error.
<i>internal_query</i>	<i>internal_query</i>	<i>text</i>	Internal query that led to the error, if any.
<i>internal_query_pos</i>	<i>internal_query_pos</i>	<i>integer</i>	Character count of the error position therein.
<i>context</i>	<i>context</i>	<i>text</i>	Error context.
<i>query</i>	<i>query</i>	<i>text</i>	User query that led to the error. For errors only.
<i>query_pos</i>	<i>query_pos</i>	<i>integer</i>	Character count of the error position therein. For errors only.
<i>location</i>	<i>location</i>	<i>text</i>	Location of the error in the source code. The location field is populated if <i>log_error_verbosity</i> is set to verbose.
<i>application_name</i>	<i>event/application_name</i>	<i>text</i>	Name of the application from which the statement was executed, for example, <i>psql.bin</i> .

continues on next page

Table 5 – continued from previous page

Field	XML element/attribute	Data type	Description
<i>backend_type</i>	<i>event/backend_type</i>	<i>text</i>	The <i>backend_type</i> corresponds to what <i>pg_stat_activity.backend_type</i> shows and is added as a column to the csv log.
<i>leader_pid</i>	<i>event/leader_pid</i>	<i>integer</i>	Process ID of leader for active parallel workers.
<i>query_id</i>	<i>event/query_id</i>	<i>long</i>	Identifier of this backend's most recent query.
<i>command_tag</i>	<i>event/command_tag</i>	<i>text</i>	SQL command of the statement.
<i>audit_tag</i>	<i>event/audit_tag</i>	<i>text</i>	Value specified by the <i>audit_tag</i> parameter in the configuration file.
<i>type</i>	<i>event/type</i>	<i>text</i>	Determines the audit <i>event_type</i> to identify messages in the log.

The following examples are generated in the CSV and XML formats.

The non-default audit settings in the `postgresql.conf` file are as follows:

```
logging_collector = on
edb_audit = csv
edb_audit_connect = all
edb_audit_disconnect = all
edb_audit_statement = ddl, dml, select, error
edb_audit_tag = edbaudit
```

The `edb_audit` parameter is changed to `xml` when generating the XML format.

The following is the audited session:

```
$ psql edb enterisedb
Password for user enterisedb:
psql.bin (14.0.0)
Type "help" for help.

edb=# CREATE SCHEMA edb;
CREATE SCHEMA
edb=# SET search_path TO edb;
SET
edb=# CREATE TABLE dept (
edb(#      deptno          NUMBER(2) NOT NULL CONSTRAINT dept_pk PRIMARY KEY,
edb(#      dname           VARCHAR2(14) CONSTRAINT dept_dname_uq UNIQUE,
edb(#      loc             VARCHAR2(13)
edb(# );
CREATE TABLE
edb=# INSERT INTO dept VALUES (10,ACCOUNTING,NEW YORK);
INSERT 0 1
edb=# UPDATE department SET loc = BOSTON WHERE deptno = 10;
ERROR:  relation "department" does not exist
LINE 1: UPDATE department SET loc = BOSTON WHERE deptno = 10;
          ^
edb=# UPDATE dept SET loc = BOSTON WHERE deptno = 10;
```

(continues on next page)

(continued from previous page)

```

UPDATE 1
edb=# SELECT * FROM dept;
__OUTPUT__
 deptno |   dtype   | loc
-----+-----+-----
      10 | ACCOUNTING | BOSTON
(1 row)

edb=# \q

```

CSV audit log file

The following is the CSV format of the audit log file. (Each audit log entry was split and displays across multiple lines. A blank line was inserted between the audit log entries for visual clarity.)

```

2022-12-14 12:19:01.035 UTC,"enterprisedb","edb",9290,"[local]",
6399bf35.244a,1,"authentication",2022-12-14 12:19:01 UTC,4/19,0,
AUDIT,000000,"connection authorized: user=enterprisedb database=edb",
,,,,,,,,,"client backend",,0,"","edbaudit","connect"

2022-12-14 12:19:12.599 UTC,"enterprisedb","edb",9293,"[local]",
6399bf40.244d,1,"authentication",2022-12-14 12:19:12 UTC,5/1,0,
AUDIT,000000,"connection authorized: user=enterprisedb database=edb",
,,,,,,,,,"client backend",,0,"","edbaudit","connect"

2022-12-14 12:19:21.351 UTC,"enterprisedb","edb",9293,"[local]",
6399bf40.244d,2,"idle",2022-12-14 12:19:12 UTC,5/3,0,AUDIT,000000,
"statement: CREATE SCHEMA edb;",,,,,,,,,,"psql","client backend",,
0,"CREATE SCHEMA","edbaudit","create"

2022-12-14 12:19:27.817 UTC,"enterprisedb","edb",9293,"[local]",
6399bf40.244d,3,"idle",2022-12-14 12:19:12 UTC,5/4,0,AUDIT,000000,
"statement: CREATE SCHEMA edb;",,,,,,,,,,"psql","client backend",,
0,"CREATE SCHEMA","edbaudit","create"

2022-12-14 12:19:27.820 UTC,"enterprisedb","edb",9293,"[local]",
6399bf40.244d,4,"CREATE SCHEMA",2022-12-14 12:19:12 UTC,5/4,0,ERROR,
42P06,"schema ""edb"" already exists",,,,,,"CREATE SCHEMA edb;",,,
,"psql","client backend",,0,"CREATE SCHEMA","edbaudit","error"

2022-12-14 12:20:15.407 UTC,"enterprisedb","edb",9293,"[local]",
6399bf40.244d,5,"idle",2022-12-14 12:19:12 UTC,5/6,0,AUDIT,000000,
"statement: CREATE TABLE dept (
    deptno          NUMBER(2) NOT NULL CONSTRAINT dept_pk PRIMARY KEY,
    dtype           VARCHAR2(14) CONSTRAINT dept_dtype_uq UNIQUE,
    loc             VARCHAR2(13)
);",,,,,,,,,,"psql","client backend",,0,"CREATE TABLE","edbaudit",
"create"

2022-12-14 12:20:24.433 UTC,"enterprisedb","edb",9293,"[local]",
6399bf40.244d,6,"idle",2022-12-14 12:19:12 UTC,5/7,0,AUDIT,000000,

```

(continues on next page)

(continued from previous page)

```

"statement: INSERT INTO dept VALUES (10,ACCOUNTING,NEW YORK);",
,,,,,,,"psql","client backend",,0,"INSERT","edbaudit","insert"

2022-12-14 12:20:34.393 UTC,"enterprisedb","edb",9293,"[local]",
6399bf40.244d,7,"idle",2022-12-14 12:19:12 UTC,5/8,0,AUDIT,000000,
"statement: UPDATE department SET loc = BOSTON WHERE deptno = 10;",
,,,,,,,"psql","client backend",,0,"UPDATE","edbaudit","update"

2022-12-14 12:20:34.394 UTC,"enterprisedb","edb",9293,"[local]",
6399bf40.244d,8,"UPDATE",2022-12-14 12:19:12 UTC,5/8,0,ERROR,
42P01,"relation ""department"" does not exist",,,,,,
"UPDATE department SET loc = BOSTON WHERE deptno = 10;",8,,
"psql","client backend",,0,"UPDATE","edbaudit","error"

2022-12-14 12:20:43.721 UTC,"enterprisedb","edb",9293,"[local]",
6399bf40.244d,9,"idle",2022-12-14 12:19:12 UTC,5/9,0,AUDIT,000000,
"statement: UPDATE dept SET loc = BOSTON WHERE deptno = 10;",
,,,,,,,"psql","client backend",,0,"UPDATE","edbaudit","update"

2022-12-14 12:20:51.231 UTC,"enterprisedb","edb",9293,"[local]",
6399bf40.244d,10,"idle",2022-12-14 12:19:12 UTC,5/10,0,AUDIT,000000,
"statement: SELECT * FROM dept;",,,,,,,,"psql","client backend",,0,
"SELECT","edbaudit","select"

2022-12-14 12:20:53.940 UTC,"enterprisedb","edb",9293,"[local]",
6399bf40.244d,11,"idle",2022-12-14 12:19:12 UTC,,0,AUDIT,000000,
"disconnection: session time: 0:01:41.344 user=enterprisedb database=edb
host=[local]",,,,,,,,"psql","client backend",,0,"","edbaudit","disconnect"

```

XML audit log file

The following is the XML format of the audit log file. (The output was formatted for visual clarity.)

```

<event user="amul" database="edb" process_id="110405"
  remote_host="[local]" session_id="63e1f4cf.1af45"
  session_line_num="1" process_status="idle"
  session_start_time="2023-02-07 12:20:56.920 IST"
  log_time="2023-02-07 12:20:55 IST" virtual_transaction_id="4/22"
  type="create" command_tag="CREATE ROLE" application_name="psql"
  backend_type="client backend" query_id="0">
  <error_severity>AUDIT</error_severity>
  <message>statement: create user edb superuser;</message>
</event>
<event user="edb" database="postgres" process_id="110424"
  remote_host="[local]" session_id="63e1f4d4.1af58"
  session_line_num="1" process_status="idle"
  session_start_time="2023-02-07 12:21:00.561 IST"
  log_time="2023-02-07 12:21:00 IST" virtual_transaction_id="4/25"
  type="set" command_tag="SET" application_name="pg_regress"
  backend_type="client backend" query_id="0">
  <error_severity>AUDIT</error_severity>

```

(continues on next page)

(continued from previous page)

```

    <message>statement: SET client_min_messages = warning</message>
</event>
<event user="edb" database="postgres" process_id="110424"
  remote_host="[local]" session_id="63e1f4d4.1af58"
  session_line_num="2" process_status="idle"
  session_start_time="2023-02-07 12:21:00.561 IST"
  log_time="2023-02-07 12:21:00 IST" virtual_transaction_id="4/26"
  type="drop" command_tag="DROP DATABASE" application_name="pg_regress"
  backend_type="client backend" query_id="0">
  <error_severity>AUDIT</error_severity>
  <message>statement: DROP DATABASE IF EXISTS &quot;regression&quot;;</message>
</event>
<event user="edb" database="postgres" process_id="110426"
  remote_host="[local]" session_id="63e1f4d4.1af5a"
  session_line_num="1" process_status="idle"
  session_start_time="2023-02-07 12:21:00.568 IST"
  log_time="2023-02-07 12:21:00 IST" virtual_transaction_id="4/29"
  type="create" command_tag="CREATE DATABASE" application_name="pg_regress"
  backend_type="client backend" query_id="0">
  <error_severity>AUDIT</error_severity>
  <message>statement: CREATE DATABASE &quot;regression&quot;; TEMPLATE=template0</
  message>
</event>
<event user="edb" database="postgres" process_id="110426"
  remote_host="[local]" session_id="63e1f4d4.1af5a"
  session_line_num="2" process_status="idle"
  session_start_time="2023-02-07 12:21:00.983 IST"
  log_time="2023-02-07 12:21:00 IST" virtual_transaction_id="4/30"
  type="alter" command_tag="ALTER DATABASE" application_name="pg_regress"
  backend_type="client backend" query_id="0">
  <error_severity>AUDIT</error_severity>
  <message>statement: ALTER DATABASE &quot;regression&quot;; SET lc_messages TO &apos;C&
  apus&apos;;ALTER DATABASE &quot;regression&quot;; SET lc_monetary TO &apos;C&apos;C&apos;;ALTER
  DATABASE &quot;regression&quot;; SET lc_numeric TO &apos;C&apos;C&apos;;ALTER DATABASE &quot;
  regression&quot;; SET lc_time TO &apos;C&apos;C&apos;;ALTER DATABASE &quot;regression&quot;;
  SET bytea_output TO &apos;hex&apos;;ALTER DATABASE &quot;regression&quot;; SET timezone_
  abbreviations TO &apos;Default&apos;;</message>
</event>
<event user="edb" database="regression" process_id="110429"
  remote_host="[local]" session_id="63e1f4d5.1af5d"
  session_line_num="1" process_status="idle"
  session_start_time="2023-02-07 12:21:01.152 IST"
  log_time="2023-02-07 12:21:01 IST" virtual_transaction_id="4/33"
  type="set" command_tag="SET" application_name="pg_regress/test_setup"
  backend_type="client backend" query_id="0">
  <error_severity>AUDIT</error_severity>
  <message>statement: SET synchronous_commit = on;</message>
</event>
<event user="enterprisedb" database="edb" process_id="5942" remote_host=
"[local]"
  session_id="5ec7ac4d.1735" session_line_num="8" process_status="idle"
  session_start_time="2023-02-07 16:11:17 IST" log_time="2023-02-07

```

(continues on next page)

(continued from previous page)

```

16:12:45.471 IST" virtual_transaction_id="4/27" type="select"
command_tag="SELECT" audit_tag="edbaudit" application_name="psql"
backend_type="parallel worker" leader_pid="5940" query_id="0">
  <error_severity>AUDIT</error_severity>
  <message>statement: SELECT * FROM dept;</message>
</event>
<event process_id="112399" session_id="63elf4e2.1b70f" session_line_num="1"
session_start_time="2023-02-07 12:21:14.150 IST"
log_time="2023-02-07 12:21:14 IST" virtual_transaction_id="5/2012"
type="error" sql_state_code="42601" command_tag="SELECT"
application_name="pg_regress/timestamptz" backend_type="parallel worker"
leader_pid="112317" query_id="0">
  <error_severity>ERROR</error_severity>
  <message>date format not recognized</message>
  <query>SELECT to_char(d1, E'HH:MI:SS is'
HH:MI:SS 'text between quote marks'
FROM TIMESTAMPTZ_TBL;</query>
</event>
<event process_id="112402" session_id="63elf4e2.1b712" session_line_num="1"
session_start_time="2023-02-07 12:21:14.156 IST" log_time="2023-02-07
12:21:14 IST" virtual_transaction_id="5/2015" type="error"
sql_state_code="42601" command_tag="SELECT"
application_name="pg_regress/timestamptz" backend_type="parallel worker"
leader_pid="112317" query_id="0">
  <error_severity>ERROR</error_severity>
  <message>date format not recognized</message>
  <query>SELECT to_char(d1, 'HH24--text--MI--text--SS'
FROM TIMESTAMPTZ_TBL;</query>
</event>

```

3.5.5 Using error codes to filter audit logs

EDB Postgres Advanced Server includes an extension that you can use to exclude from the EDB Postgres Advanced Server log files entries that include a user-specified error code. To filter audit log entries, you must first enable the extension by modifying the `postgresql.conf` file. Add the following value to those specified in the `shared_preload_libraries` parameter:

```
$libdir/edb_filter_log
```

Then, use the `edb_filter_log.errcodes` parameter to specify any error codes you want to omit from the log files:

```
edb_filter_log.errcode = error_code
```

Where `error_code` specifies one or more error codes that you want to omit from the log file. Provide multiple error codes in a comma-delimited list.

For example, if `edb_filter_log` is enabled, and `edb_filter_log.errcode` is set to `'23505,23502,22012'`, any log entries that return one of the following SQLSTATE errors are omitted from the log file:

23505 (for violating a unique constraint)

23502 (for violating a not-null constraint)

22012 (for dividing by zero)

For a complete list of the error codes supported by EDB Postgres Advanced Server audit log filtering, see the [Postgres core documentation](#).

3.5.6 Using command tags to filter audit logs

Each entry in the log file contains a *command tag*, except for entries that display an error message. A command tag is the SQL command executed for that log entry. The command tag makes it possible to use tools later to scan the log file to find entries related to a SQL command.

This example is in XML form, formatted for easier review. The command tag is displayed as the `command_tag` attribute of the event element with values `CREATE ROLE`, `ALTER ROLE`, and `DROP ROLE`.

```
<event user="enterprisedb" database="edb" process_id="64234" remote_host=
"[local]"
  session_id="5ecbc7e6.faea" session_line_num="2" process_status="idle"
  session_start_time="2020-05-25 18:58:06 IST" log_time="2020-05-25
18:58:21.147 IST"
  virtual_transaction_id="4/30" type="create" command_tag="CREATE ROLE"
  application_name="psql" backend_type="client backend">
  <error_severity>AUDIT</error_severity>
  <message>statement: CREATE ROLE newuser WITH LOGIN;</message>
</event>
<event user="enterprisedb" database="edb" process_id="64234" remote_host=
"[local]"
  session_id="5ecbc7e6.faea" session_line_num="3" process_status="idle"
  session_start_time="2020-05-25 18:58:06 IST" log_time="2020-05-25
18:58:34.142 IST"
  virtual_transaction_id="4/31" type="error" sql_state_code="42601"
  application_name="psql" backend_type="client backend">
  <error_severity>ERROR</error_severity>
  <message>unrecognized role option &quot;super&quot;;</message>
  <query>ALTER ROLE newuser WITH SUPER USER;</query>
  <query_pos>25</query_pos>
</event>
<event user="enterprisedb" database="edb" process_id="64234" remote_host=
"[local]"
  session_id="5ecbc7e6.faea" session_line_num="4" process_status="idle"
  session_start_time="2020-05-25 18:58:06 IST" log_time="2020-05-25
18:58:44.680 IST"
  virtual_transaction_id="4/32" type="alter" command_tag="ALTER ROLE"
  application_name="psql" backend_type="client backend">
  <error_severity>AUDIT</error_severity>
  <message>statement: ALTER ROLE newuser WITH SUPERUSER;</message>
</event>
<event user="enterprisedb" database="edb" process_id="64234" remote_host=
"[local]"
  session_id="5ecbc7e6.faea" session_line_num="5" process_status="idle"
  session_start_time="2020-05-25 18:58:06 IST" log_time="2020-05-25
18:58:58.173 IST"
  virtual_transaction_id="4/33" type="drop" command_tag="DROP ROLE"
  application_name="psql" backend_type="client backend">
  <error_severity>AUDIT</error_severity>
  <message>statement: DROP ROLE newuser;</message>
```

(continues on next page)

(continued from previous page)

</event>

The following is the same example in CSV form. The command tag is the next-to-last column of each entry. In the listing, the column that appears empty (""), is the value of the `edb_audit_tag` parameter, if provided.

(Each audit log entry was split and displays across multiple lines. A blank line was inserted between the audit log entries for visual clarity.)

```
2020-05-25 19:09:32.105 IST,"enterprisedb","edb",77212,"[local]",5ecbca7b.
12d9c,2,"idle",2020-05-25 19:09:07 IST,4/30,0,AUDIT,00000,"statement: CREATE
ROLE newuser WITH LOGIN;",,,,,,,,,,"psql","client backend","CREATE ROLE","",
"create"

2020-05-25 19:09:50.975 IST,"enterprisedb","edb",77212,"[local]",5ecbca7b.
12d9c,3,"idle",2020-05-25 19:09:07 IST,4/31,0,ERROR,42601,"unrecognized role
option ""super""",,,,,,"ALTER ROLE newuser WITH SUPER USER;",25,,,"psql",
"client backend","", "", "error"

2020-05-25 19:10:04.128 IST,"enterprisedb","edb",77212,"[local]",5ecbca7b.
12d9c,4,"idle",2020-05-25 19:09:07 IST,4/32,0,AUDIT,00000,"statement: ALTER
ROLE newuser WITH SUPERUSER;",,,,,,,,,,"psql","client backend","ALTER ROLE","",
"alter"

2020-05-25 19:10:15.959 IST,"enterprisedb","edb",77212,"[local]",5ecbca7b.
12d9c,5,"idle",2020-05-25 19:09:07 IST,4/33,0,AUDIT,00000,"statement: DROP
ROLE newuser;",,,,,,,,,,"psql","client backend","DROP ROLE","", "drop"
```

3.5.7 Redacting passwords in audit logs

You can use the GUC parameter `edb_filter_log.redact_password_commands` under the `postgresql.conf` file to redact stored passwords in the log file.

The syntax is:

```
{CREATE|ALTER} {USER|ROLE|GROUP} identifier { [WITH] [ENCRYPTED] PASSWORD
nonempty_string_literal | IDENTIFIED BY { nonempty_string_literal |
bareword } } [ REPLACE { nonempty_string_literal | bareword } ]
```

To enable password redaction, you must first enable the parameter by modifying the `postgresql.conf` file. Add the following value to those specified in the `shared_preload_libraries` parameter:

```
$libdir/edb_filter_log
```

Then, set `edb_filter_log.redact_password_commands` to `true` :

```
edb_filter_log.redact_password_commands = true
```

After modifying the `postgresql.conf` file, you must restart the server for the changes to take effect.

Examples

When the following statement is logged by `log_statement`, the server redacts the password to `x`. For example, this command is added to the log file:

```
CREATE USER carol WITH login PASSWORD 1safepwd;
```

It appears as:

```
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,000000,"statement: CREATE USER carol with login_
→PASSWORD x;",,,,,,,,,,"psql","client backend","CREATE ROLE","", "create"
```

When the following statement is logged by `log_statement`, the server identifies the new password, replace, and redact the password to `x`. For example, this command is added to the log file:

```
ALTER USER carol IDENTIFIED BY new_pass REPLACE 1safepwd;
```

It appears as:

```
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,000000,"statement: ALTER USER carol IDENTIFIED BY x_
→REPLACE x;",,,,,,,,,,"psql","client backend","ALTER ROLE","", "alter"
```

The statement that includes a password is logged. The server redacts the password text to `x`. When the statement is logged as context for some other message, the server omits the statement from the context.

3.5.8 Audit log archiving

EDB audit log archiving enables database administrators to control the space consumed by the audit log directory and helps manage the audit log files. The Audit Archiver is responsible for the compression, execution, and expiration of log files with `edb_audit_archiver_filename_prefix` present in the audit directory. The `edb_audit_archiver_timeout` parameter triggers the compression or expiration of audit log files at an appropriate time. For more information about audit archiving configuration parameters, see [Audit logging configuration parameters](#).

The audit archiver helps to:

- Prepare a list of log files from the audit log directory for compression.
- Determine the log files for compression based on the parameters `edb_audit_archiver_compress_time_limit` and `edb_audit_archiver_compress_size_limit`.
- Perform compression of the log files by specifying the compression command based on the `edb_audit_archiver_compress_command` parameter.
- Determine the log files to remove based on the `edb_audit_archiver_expire_time_limit` and `edb_audit_archiver_expire_size_limit` parameter.
- Execute the expiration command specified in the `edb_audit_archiver_expire_command` parameter to remove the log files.

To rotate out the older audit log files, you can set the log file rotation day when the new file is created. To do so, set the parameter `edb_audit_rotation_day` to the desired value. The audit log records are overwritten on a first-in, first-out basis if space isn't available for more audit log records.

To configure EDB Postgres Advanced Server to enable compression and expiration of the log files:

1. Enable audit log archiving by setting the `edb_audit_archiver` parameter to `on` in the `postgresql.conf` file.
2. To enable compression of log files, set the parameter `edb_audit_archiver_compress_size_limit` and `edb_audit_archiver_compress_time_limit` to the values you want.
3. To enable expiration of log files, set the parameter `edb_audit_archiver_expire_size_limit` and `edb_audit_archiver_expire_time_limit` to the values you want.
4. To enable both compression and expiration, set the parameters `edb_audit_archiver_compress_size_limit` , `edb_audit_archiver_compress_time_limit` , `edb_audit_archiver_expire_size_limit` , `edb_audit_archiver_expire_time_limit` , and `edb_audit_archiver_expire_command` to the values you want.

The following is an example of the configuration parameter settings in the `postgresql.conf` file.

Setting the `edb_audit_archiver` parameter in the configuration file affects the entire database cluster. The database cluster is established with `edb_audit_archiver` set to `on` , as shown in the `postgresql.conf` file. The audit log file is generated in CSV format based on the setting of the `edb_audit` configuration parameter.

```
logging_collector = on
edb_audit = csv
edb_audit_archiver = on # (Change requires restart)
```

Examples

In this example, `edb_audit_archiver` , `edb_audit_archiver_compress_size_limit` , and `edb_audit_archiver_compress_time_limit` are set to enable compression of the audit log files greater than 10MB.

```
edb_audit_archiver = on
edb_audit_archiver_compress_size_limit = 10
edb_audit_archiver_compress_time_limit = 0
```

Before compression, the audit log file appears as follows:

```
- rw-----. 1 enterprisedb edb 2097276 Jun 4 11:42 audit-20200811_114237.csv <==
↳ Shows non-compressed audit log files
- rw-----. 1 enterprisedb edb 2097278 Jun 4 11:42 audit-20200811_114243.csv
- rw-----. 1 enterprisedb edb 2097289 Jun 4 11:42 audit-20200811_114245.csv
- rw-----. 1 enterprisedb edb 2097330 Jun 4 11:42 audit-20200811_114246.csv
- rw-----. 1 enterprisedb edb 2097343 Jun 4 11:42 audit-20200811_114248.csv
- rw-----. 1 enterprisedb edb 2097338 Jun 4 11:42 audit-20200811_114249.csv
```

The `edb_audit_archiver` parameter checks the log files, excluding the latest file. It retains at least 10MB of log files in the audit log directory and compresses the remaining log files. The `.gz` specifies the name of an already compressed log file. After compression, the audit log file appears as follows:

```
- rw-----. 1 enterprisedb edb 64119 Jun 4 11:42 audit-20200811_114237.csv.gz <==
↳ Shows compressed audit log files
- rw-----. 1 enterprisedb edb 64323 Jun 4 11:42 audit-20200811_114243.csv.gz
- rw-----. 1 enterprisedb edb 64091 Jun 4 11:42 audit-20200811_114245.csv.gz
- rw-----. 1 enterprisedb edb 64152 Jun 4 11:42 audit-20200811_114246.csv.gz
- rw-----. 1 enterprisedb edb 2097343 Jun 4 11:42 audit-20200811_114248.csv <==
↳ Shows non-compressed audit log files
- rw-----. 1 enterprisedb edb 2097338 Jun 4 11:42 audit-20200811_114249.csv
```

In this example, `edb_audit_archiver` , `edb_audit_archiver_expire_size_limit` , and `edb_audit_archiver_expire_time_limit` are set to enable expiration of the audit log files older than one hour.

```
edb_audit_archiver = on
edb_audit_archiver_expire_size_limit = 0
edb_audit_archiver_expire_time_limit = 3600
```

Before compression, the audit log file appears as follows:

```
- rw----- . 1 enterprisedb edb 2097367 Jun 4 13:23 audit-20200811_132352.csv <==
↳ Shows non-compressed audit log files scheduled for expiration/removal
- rw----- . 1 enterprisedb edb 2097395 Jun 4 13:24 audit-20200811_132353.csv
- rw----- . 1 enterprisedb edb 2097328 Jun 4 14:35 audit-20200811_143438.csv
- rw----- . 1 enterprisedb edb 2097276 Jun 4 14:35 audit-20200811_143502.csv
- rw----- . 1 enterprisedb edb 2097211 Jun 4 14:35 audit-20200811_143503.csv
- rw----- . 1 enterprisedb edb 2097152 Jun 4 14:35 audit-20200811_143504.csv
```

The `edb_audit_archiver` parameter checks the log files, excluding the latest file. It removes the log files older than one hour. After expiration, the audit log file appears as follows:

```
- rw----- . 1 enterprisedb edb 2097328 Jun 4 14:35 audit-20200811_143438.csv
- rw----- . 1 enterprisedb edb 2097276 Jun 4 14:35 audit-20200811_143502.csv
- rw----- . 1 enterprisedb edb 2097211 Jun 4 14:35 audit-20200811_143503.csv
- rw----- . 1 enterprisedb edb 2097152 Jun 4 14:35 audit-20200811_143504.csv
```

In this example, `edb_audit_archiver` , `edb_audit_archiver_compress_size_limit` , `edb_audit_archiver_compress_time_limit` , `edb_audit_archiver_expire_size_limit` , `edb_audit_archiver_expire_time_limit` , and `edb_audit_archiver_expire_command` are set to enable both compression and expiration of the audit log files.

```
edb_audit_archiver = on
edb_audit_archiver_compress_size_limit = 4
edb_audit_archiver_compress_time_limit = 0
edb_audit_archiver_expire_size_limit = 5
edb_audit_archiver_expire_time_limit = 0
edb_audit_archiver_expire_command = cp %p /home/edb_audit/backup-log/
```

Before compression, the audit log file appears as follows:

```
- rw----- . 1 enterprisedb edb 2097522 Jun 4 13:02 audit-20200811_125816.csv <==
↳ Shows non-compressed audit log files ready to be removed/expired
- rw----- . 1 enterprisedb edb 2097510 Jun 4 13:02 audit-20200811_130215.csv
- rw----- . 1 enterprisedb edb 2097336 Jun 4 13:10 audit-20200811_130947.csv
- rw----- . 1 enterprisedb edb 2097271 Jun 4 13:10 audit-20200811_131034.csv
- rw----- . 1 enterprisedb edb 2097246 Jun 4 13:10 audit-20200811_131035.csv
- rw----- . 1 enterprisedb edb 2097289 Jun 4 13:10 audit-20200811_131036.csv
```

The `edb_audit_archiver` parameter checks the log files, excluding the latest file. It retains at least 4MB of log files in the audit log directory and compresses the remaining log files. While checking the log files for expiration, `edb_audit_archiver` retains at least 5MB of log files in the audit log directory and removes the remaining log files. After compression and expiration, the audit log file appears as follows:

```
- rw----- . 1 enterprisedb edb 63854 Jun 4 13:10 audit-20200811_131034.csv.gz
- rw----- . 1 enterprisedb edb 63513 Jun 4 13:10 audit-20200811_131035.csv.gz
- rw----- . 1 enterprisedb edb 63738 Jun 4 13:10 audit-20200811_131036.csv.gz
```

The expiration command is specified as `edb_audit_archiver_expire_command = 'cp %p /home/edb_audit/backup-log/'` in the `postgresql.conf` file. The `edb_audit_archiver` executes this command and copies the log files to a backup log directory before deleting them from the audit log directory. After compression and expiration, the backup log directory appears as follows:

```
- rw----- . 1 enterprisedb edb 2097522 Jun 4 13:02 audit-20200811_125816.csv.gz
- rw----- . 1 enterprisedb edb 2097510 Jun 4 13:02 audit-20200811_130215.csv.gz
- rw----- . 1 enterprisedb edb 2097336 Jun 4 13:10 audit-20200811_130947.csv.gz
```

3.5.9 Object auditing

Object-level auditing allows selective auditing of objects for specific data manipulation language (DML) statements, such as `SELECT`, `UPDATE`, `DELETE`, and `INSERT`, on a given table. Object-level auditing also lets you include or exclude specific groups by specifying `(@)` or `(-)` with the `edb_audit_statement` parameter. For more information about DML statements, see [Selecting SQL statements to audit](#).

Use the following syntax to specify an `edb_audit_statement` parameter value for `SELECT`, `UPDATE`, `DELETE`, or `INSERT` statements:

```
{ select | update | delete | insert }{@ | -}groupname
```

Example

In this example, `edb_audit_connect` and `edb_audit_statement` are set with the following non-default values:

```
logging_collector = on
edb_audit_connect = all
edb_audit = csv
edb_audit_statement = select, insert, update, delete
```

The SQL statements invoked by the `SELECT`, `INSERT`, `UPDATE`, and `DELETE` commands are audited.

The following is the database session that occurs:

```
$ psql edb enterprisedb
Password for user enterprisedb:
psql.bin (14.0.0)
Type "help" for help.

edb=# SHOW edb_audit_connect;
__OUTPUT__
edb_audit_connect
- -----
all
(1 row)
```

```

edb=# SHOW edb_audit_statement;
__OUTPUT__
      edb_audit_statement
-----
select, insert, update, delete
(1 row)

edb=# CREATE TABLE emp
edb=#      (empno NUMBER(4) NOT NULL,
edb=#      ename VARCHAR2(10),
edb=#      job VARCHAR2(9),
edb=#      mgr NUMBER(4),
edb=#      hiredate DATE,
edb=#      sal NUMBER(7, 2),
edb=#      comm NUMBER(7, 2),
edb=#      deptno NUMBER(2));
CREATE TABLE

edb=# ALTER TABLE emp SET (edb_audit_group = low_security);
ALTER TABLE

edb=# CREATE TABLE dept
edb=#      (deptno NUMBER(2),
edb=#      dname VARCHAR2(14),
edb=#      loc VARCHAR2(13) ) with (edb_audit_group = low_security);
CREATE TABLE

edb=# CREATE TABLE bonus
edb=#      (ename VARCHAR2(10),
edb=#      job VARCHAR2(9),
edb=#      sal NUMBER,
edb=#      comm NUMBER) with (edb_audit_group = high_security);
CREATE TABLE

edb=# CREATE TABLE sal
edb=#      (grade NUMBER,
edb=#      losal NUMBER,
edb=#      hisal NUMBER) with (edb_audit_group = high_security);
CREATE TABLE

edb=# SET edb_audit_statement = select@low_security@high_security, insert@high_security-
↳ low_security, update-low_security@high_security, delete@low_security-high_security;
SET

```

```

edb=# SELECT reloptions FROM pg_class WHERE relname IN(emp, dept, bonus, sal);
__OUTPUT__
      reloptions
-----
{edb_audit_group=low_security}
{edb_audit_group=low_security}
{edb_audit_group=high_security}
{edb_audit_group=high_security}
(4 rows)

```

```

edb=# SELECT setting FROM pg_settings WHERE name = edb_audit_statement;
__OUTPUT__
              setting
-----
select@low_security@high_security, insert@high_security-low_security, update-low_
security@high_security, delete@low_security-high_security
(1 row)

```

```

edb=# SELECT * FROM emp;
__OUTPUT__
 empno | ename | job | mgr | hiredate | sal | comm | deptno
-----+-----+-----+-----+-----+-----+-----+-----
(0 rows)

```

```

edb=# SELECT * FROM dept;
__OUTPUT__
 deptno | dname | loc
-----+-----+-----
(0 rows)

```

```

edb=# SELECT * FROM bonus;
__OUTPUT__
 ename | job | sal | comm
-----+-----+-----+-----
(0 rows)

```

```

edb=# SELECT * FROM sal;
__OUTPUT__
 grade | losal | hisal
-----+-----+-----
(0 rows)

```

```

edb=# INSERT INTO emp VALUES (7369, SMITH, CLERK, 7902, TO_DATE(17-DEC-1980, DD-MON-
→YYYY), 800, NULL, 10);
INSERT 0 1
edb=# INSERT INTO emp VALUES (7788, SCOTT, ANALYST, 7566, TO_DATE(09-DEC-1982, DD-MON-
→YYYY), 3000, NULL, 20);
INSERT 0 1
edb=# INSERT INTO emp VALUES (7566, JONES, MANAGER, 7839, TO_DATE(2-APR-1981, DD-MON-
→YYYY), 2975, NULL, 30);
INSERT 0 1

edb=# INSERT INTO dept VALUES (10, ACCOUNTING, NEW YORK);
INSERT 0 1
edb=# INSERT INTO dept VALUES (20, RESEARCH, DALLAS);
INSERT 0 1
edb=# INSERT INTO dept VALUES (30, SALES, CHICAGO);
INSERT 0 1
edb=# INSERT INTO dept VALUES (40, OPERATIONS, BOSTON);
INSERT 0 1

```

(continues on next page)

(continued from previous page)

```

edb=# INSERT INTO bonus VALUES (SMITH, CLERK, 800, NULL);
INSERT 0 1
edb=# INSERT INTO bonus VALUES (SCOTT, ANALYST, 3000, NULL);
INSERT 0 1
edb=# INSERT INTO bonus VALUES (JONES, MANAGER, 2975, 300);
INSERT 0 1

edb=# INSERT INTO sal VALUES (3, 800, 1500);
INSERT 0 1
edb=# INSERT INTO sal VALUES (2, 2975, 4000);
INSERT 0 1
edb=# INSERT INTO sal VALUES (1, 3000, 5000);
INSERT 0 1

edb=# UPDATE emp SET sal = 5500 WHERE deptno = 10;
UPDATE 1
edb=# UPDATE dept SET loc = BEDFORD WHERE deptno = 20;
UPDATE 1
edb=# UPDATE bonus SET sal = 4000 WHERE ename = SMITH;
UPDATE 1
edb=# UPDATE sal SET losal = 5000;
UPDATE 3

edb=# DELETE FROM emp WHERE deptno = 10;
DELETE 1
edb=# DELETE FROM dept WHERE deptno = 20;
DELETE 1
edb=# DELETE FROM bonus WHERE sal = 4000;
DELETE 1
edb=# DELETE FROM sal WHERE losal = 5000;
DELETE 3

```

Setting the `edb_audit_statement` parameter to `'select@low_security@high_security, insert@high_security-low_security, update-low_security@high_security, delete@low_security-high_security'` for the `enterprisedb` user and `edb` database allows auditing of `SELECT`, `INSERT`, `UPDATE` or `DELETE` statements including (@) and excluding – for a group in the audit log file.

For a table in the log file:

- `select@low_security@high_security` audits `SELECT` statements of the `low_security` and `high_security` audit groups.
- `insert@high_security-low_security` audits `INSERT` statements of `high_security` and excludes the insert statements of `low_security` audit group.
- `update-low_security@high_security` audits `UPDATE` statements of `high_security` and excludes the update statements of the `low_security` audit group.
- `delete@low_security-high_security` audits `DELETE` statements of `low_security` and excludes the delete statements of `high_security` audit group for a table in the log file.

The resulting audit log file contains the following. (Each audit log entry was split and displays across multiple lines. A blank line was inserted between the audit log entries for visual clarity.)

```

2021-07-20 01:45:27.077 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,1,"authentication",2021-07-20 01:45:27 IST,5/1,0,AUDIT,00000,"connection_
↪authorized: user=enterprisedb
database=edb",,,,,,,,,,"","client backend",,,,,,,,,,"connect"

2021-07-20 01:50:40.125 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,2,"SELECT",2021-07-20 01:45:27 IST,5/13,0,AUDIT,00000,"statement: SELECT * FROM_
↪emp;",,,,,,,,,,
"psql","client backend",,"SELECT",,,,,,"select"

2021-07-20 01:50:53.778 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,3,"SELECT",2021-07-20 01:45:27 IST,5/14,0,AUDIT,00000,"statement: SELECT * FROM_
↪dept;",,,,,,,,,,
"psql","client backend",,"SELECT",,,,,,"select"

2021-07-20 01:51:05.306 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,4,"SELECT",2021-07-20 01:45:27 IST,5/15,0,AUDIT,00000,"statement: SELECT * FROM_
↪bonus;",,,,,,,,,,
"psql","client backend",,"SELECT",,,,,,"select"

2021-07-20 01:51:14.874 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,5,"SELECT",2021-07-20 01:45:27 IST,5/16,0,AUDIT,00000,"statement: SELECT * FROM_
↪sal;",,,,,,,,,,
"psql","client backend",,"SELECT",,,,,,"select"

2021-07-20 01:52:53.413 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,6,"INSERT",2021-07-20 01:45:27 IST,5/24,0,AUDIT,00000,"statement: INSERT INTO_
↪bonus VALUES (SMITH, CLERK, 800, NULL);
",,,,,,,,,,"psql","client backend",,"INSERT",,,,,,"insert"

2021-07-20 01:53:02.945 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,7,"INSERT",2021-07-20 01:45:27 IST,5/25,0,AUDIT,00000,"statement: INSERT INTO_
↪bonus VALUES (SCOTT, ANALYST, 3000, NULL);
",,,,,,,,,,"psql","client backend",,"INSERT",,,,,,"insert"

2021-07-20 01:53:12.064 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,8,"INSERT",2021-07-20 01:45:27 IST,5/26,0,AUDIT,00000,"statement: INSERT INTO_
↪bonus VALUES (JONES, MANAGER, 2975, 300);
",,,,,,,,,,"psql","client backend",,"INSERT",,,,,,"insert"

2021-07-20 01:53:23.836 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,9,"INSERT",2021-07-20 01:45:27 IST,5/27,0,AUDIT,00000,"statement: INSERT INTO sal_
↪VALUES (3, 800, 1500);
",,,,,,,,,,"psql","client backend",,"INSERT",,,,,,"insert"

2021-07-20 01:53:33.280 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,10,"INSERT",2021-07-20 01:45:27 IST,5/28,0,AUDIT,00000,"statement: INSERT INTO sal_
↪VALUES (2, 2975, 4000);
",,,,,,,,,,"psql","client backend",,"INSERT",,,,,,"insert"

2021-07-20 01:53:42.388 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,11,"INSERT",2021-07-20 01:45:27 IST,5/29,0,AUDIT,00000,"statement: INSERT INTO sal_
↪VALUES (1, 3000, 5000);

```

(continues on next page)

(continued from previous page)

```

",,,,,,,,,,"psql","client backend",,"INSERT",,"","insert"

2021-07-20 01:54:17.126 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,12,"UPDATE",2021-07-20 01:45:27 IST,5/32,0,AUDIT,00000,"statement: UPDATE bonus.
↳SET sal = 4000 WHERE ename = SMITH;
",,,,,,,,,,"psql","client backend",,"UPDATE",,"","update"

2021-07-20 01:54:34.344 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,13,"UPDATE",2021-07-20 01:45:27 IST,5/33,0,AUDIT,00000,"statement: UPDATE sal SET.
↳losal = 5000;
",,,,,,,,,,"psql","client backend",,"UPDATE",,"","update"

2021-07-20 01:54:45.887 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,14,"DELETE",2021-07-20 01:45:27 IST,5/34,0,AUDIT,00000,"statement: DELETE FROM emp.
↳WHERE deptno = 10;
",,,,,,,,,,"psql","client backend",,"DELETE",,"","delete"

2021-07-20 01:54:54.513 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,15,"DELETE",2021-07-20 01:45:27 IST,5/35,0,AUDIT,00000,"statement: DELETE FROM.
↳dept WHERE deptno = 20;
",,,,,,,,,,"psql","client backend",,"DELETE",,"","delete"

```

3.6 Unicode collation algorithm

The *Unicode Collation Algorithm* (UCA) is a specification (*Unicode Technical Report #10*) that defines a customizable method of collating and comparing Unicode data. *Collation* means how data is sorted as with a `SELECT . . . ORDER BY` clause. *Comparison* is relevant for searches that use ranges with less than, greater than, or equal to operators.

Customizability is an important factor for various reasons such as the following.

- Unicode supports a vast number of languages. Letters that may be common to several languages may be expected to collate in different orders depending upon the language.
- Characters that appear with letters in certain languages such as accents or umlauts have an impact on the expected collation depending upon the language.
- In some languages, combinations of several consecutive characters should be treated as a single character with regards to its collation sequence.
- There may be certain preferences as to the collation of letters according to case. For example, should the lower-case form of a letter collate before the uppercase form of the same letter or vice versa.
- There may be preferences as to whether punctuation marks such as underscore characters, hyphens, or space characters should be considered in the collating sequence or should they simply be ignored as if they did not exist in the string.

Given all of these variations with the vast number of languages supported by Unicode, there is a necessity for a method to select the specific criteria for determining a collating sequence. This is what the Unicode Collation Algorithm defines.

Note: In addition, another advantage for using ICU collations (the implementation of the Unicode Collation Algorithm) is for performance. Sorting tasks, including B-tree index creation, can complete in less than half the time it takes

with a non-ICU collation. The exact performance gain depends on your operating system version, the language of your text data, and other factors.

The following sections provide a brief, simplified explanation of the Unified Collation Algorithm concepts. As the algorithm and its usage are quite complex with numerous variations, refer to the official documents cited in these sections for complete details.

3.6.1 Basic Unicode collation algorithm concepts

The official information for the Unicode Collation Algorithm is specified in *Unicode Technical Report #10*, which can be found on The Unicode Consortium website at:

<http://www.unicode.org/reports/tr10/>

The ICU – International Components for Unicode also provides much useful information. An explanation of the collation concepts can be found on their website located at:

<http://userguide.icu-project.org/collation/concepts>

The basic concept behind the Unicode Collation Algorithm is the use of multilevel comparison. This means that a number of levels are defined, which are listed as level 1 through level 5 in the following bullet points. Each level defines a type of comparison. Strings are first compared using the primary level, also called level 1.

If the order can be determined based on the primary level, then the algorithm is done. If the order cannot be determined based on the primary level, then the secondary level, level 2, is applied. If the order can be determined based on the secondary level, then the algorithm is done, otherwise the tertiary level is applied, and so on. There is typically, a final tie-breaking level to determine the order if it cannot be resolved by the prior levels.

- ****Level1-PrimaryLevelforBaseCharacters.**** The order of basic characters such as letters and digits determines the difference such as `A < B` .
- ****Level2-SecondaryLevelforAccents.**** If there are no primary level differences, then the presence or absence of accents and other such characters determine the order such as `a < á` .
- ****Level3-TertiaryLevelforCase.**** If there are no primary level or secondary level differences, then a difference in case determines the order such as `a < A` .
- ****Level4-QuaternaryLevelforPunctuation.**** If there are no primary, secondary, or tertiary level differences, then the presence or absence of white space characters, control characters, and punctuation determine the order such as `~A < A` .
- ****Level5-IdenticalLevelforTie-Breaking.**** If there are no primary, secondary, tertiary, or quaternary level differences, then some other difference such as the code point values determines the order.

3.6.2 International components for Unicode

The Unicode Collation Algorithm is implemented by open source software provided by the *International Components for Unicode* (ICU). The software is a set of C/C++ and Java libraries.

When EDB Postgres Advanced Server is used to create a collation that invokes the ICU components to produce the collation, the result is referred to as an *ICU collation*.

Locale collations

When creating a collation for a locale, a predefined ICU short form name for the given locale is typically provided.

An *ICU short form* is a method of specifying *collation attributes*, which are the properties of a collation. *Collation Attributes* provides additional information on collation attributes.

There are predefined ICU short forms for locales. The ICU short form for a locale incorporates the collation attribute settings typically used for the given locale. This simplifies the collation creation process by eliminating the need to specify the entire list of collation attributes for that locale.

The system catalog `pg_catalog.pg_icu_collate_names` contains a list of the names of the ICU short forms for locales. The ICU short form name is listed in column `icu_short_form`.

```
edb=# SELECT icu_short_form, valid_locale FROM pg_icu_collate_names ORDER BY
valid_locale;
```

```
--OUTPUT--
```

```
icu_short_form | valid_locale
```

```
-----+-----
LAF           | af
LAR           | ar
LAS           | as
LAZ           | az
LBE           | be
LBG           | bg
LBN           | bn
LBS           | bs
LBS_ZCYRL     | bs_Cyrl
LROOT         | ca
LROOT         | chr
LCS           | cs
LCY           | cy
LDA           | da
LROOT         | de
LROOT         | dz
LEE           | ee
LEL           | el
LROOT         | en
LROOT         | en_US
LEN_RUS_VPOSIX | en_US_POSIX
LEO           | eo
LES           | es
LET           | et
LFA           | fa
LFA_RAF       | fa_AF
.
```

(continues on next page)

(continued from previous page)

.

.

If needed, the default characteristics of an ICU short form for a given locale can be overridden by specifying the collation attributes to override that property. This is discussed in the next section.

Collation attributes

When creating an ICU collation, the desired characteristics of the collation must be specified. As discussed in *Locale Collations*, this can typically be done with an ICU short form for the desired locale. However, if more specific information is required, the specification of the collation properties can be done by using *collation attributes*.

Collation attributes define the rules of how characters are to be compared for determining the collation sequence of text strings. As Unicode covers a vast set of languages in numerous variations according to country, territory and culture, these collation attributes are quite complex.

For the complete, precise meaning and usage of collation attributes, see Section “Collator Naming Scheme” on the ICU – International Components for Unicode website at:

<http://userguide.icu-project.org/collation/concepts>

The following is a brief summary of the collation attributes and how they are specified using the ICU short form method

Each collation attribute is represented by an uppercase letter, which are listed in the following bullet points. The possible valid values for each attribute are given by codes shown within the parentheses. Some codes have general meanings for all attributes. **X** means to set the attribute off. **O** means to set the attribute on. **D** means to set the attribute to its default value.

- ****A-Alternate(N,S,D).**** Handles treatment of *variable* characters such as white spaces, punctuation marks, and symbols. When set to non-ignorable (N), differences in variable characters are treated with the same importance as differences in letters. When set to shifted (S), then differences in variable characters are of minor importance (that is, the variable character is ignored when comparing base characters).
- ****C-Casefirst(X,L,U,D).**** Controls whether a lowercase letter sorts before the same uppercase letter (L), or the uppercase letter sorts before the same lowercase letter (U). Off (X) is typically specified when lowercase first (L) is desired.
- ****E-Caselevel(X,O,D).**** Set in combination with the Strength attribute, the Case Level attribute is used when accents are to be ignored, but not case.
- ****F-Frenchcollation(X,O,D).**** When set to on, secondary differences (presence of accents) are sorted from the back of the string as done in the French Canadian locale.
- ****H-Hiraganaquaternary(X,O,D).**** Introduces an additional level to distinguish between the Hiragana and Katakana characters for compatibility with the JIS X 4061 collation of Japanese character strings.
- ****N-Normalizationchecking(X,O,D).**** Controls whether or not text is thoroughly normalized for comparison. Normalization deals with the issue of canonical equivalence of text whereby different code point sequences represent the same character, which then present issues when sorting or comparing such characters. Languages such as Arabic, ancient Greek, Hebrew, Hindi, Thai, or Vietnamese should be used with Normalization Checking set to on.

(continued from previous page)

- ****S-Strength(1,2,3,4,I,D).**** Maximum collation level used for comparison.
 - ↪ Influences whether accents or case are taken into account when collating or
 - ↪ comparing strings. Each number represents a level. A setting of I represents
 - ↪ identical strength (that is, level 5).
- ****T-Variabletop(hexadecimaldigits).**** Applicable only when the Alternate attribute
 - ↪ is not set to non-ignorable (N). The hexadecimal digits specify the highest
 - ↪ character sequence that is to be considered ignorable. For example, if white
 - ↪ space is to be ignorable, but visible variable characters are not to be ignorable,
 - ↪ then Variable Top set to 0020 would be specified along with the Alternate
 - ↪ attribute set to S and the Strength attribute set to 3. (The space character is
 - ↪ hexadecimal 0020. Other non-visible variable characters such as backspace, tab,
 - ↪ line feed, carriage return, etc. have values less than 0020. All visible
 - ↪ punctuation marks have values greater than 0020.)

A set of collation attributes and their values is represented by a text string consisting of the collation attribute letter concatenated with the desired attribute value. Each attribute/value pair is joined to the next pair with an underscore character as shown by the following example.

AN_CX_EX_FX_HX_NO_S3

Collation attributes can be specified along with a locale's ICU short form name to override those default attribute settings of the locale.

The following is an example where the ICU short form named LROOT is modified with a number of other collation attribute/value pairs.

AN_CX_EX_LROOT_NO_S3

In the preceding example, the Alternate attribute (A) is set to non-ignorable (N). The Case First attribute (C) is set to off (X). The Case Level attribute (E) is set to off (X). The Normalization attribute (N) is set to on (O). The Strength attribute (S) is set to the tertiary level 3. LROOT is the ICU short form to which these other attributes are applying modifications.

3.6.3 Using a collation

A newly defined ICU collation can be used anywhere the COLLATION "collation_name" clause can be used in a SQL command such as in the column specifications of the CREATE TABLE command or appended to an expression in the ORDER BY clause of a SELECT command.

The following are some examples of the creation and usage of ICU collations based on the English language in the United States (en_US.UTF8).

In these examples, ICU collations are created with the following characteristics.

Collation icu_collate_lowercase forces the lowercase form of a letter to sort before its uppercase counterpart (CL).

Collation icu_collate_uppercase forces the uppercase form of a letter to sort before its lowercase counterpart (CU).

Collation icu_collate_ignore_punct causes variable characters (white space and punctuation marks) to be ignored during sorting (AS).

Collation `icu_collate_ignore_white_sp` causes white space and other non-visible variable characters to be ignored during sorting, but visible variable characters (punctuation marks) are not ignored (AS, T0020).

```
CREATE COLLATION icu_collate_lowercase (
    LOCALE = en_US.UTF8,
    ICU_SHORT_FORM = AN_CL_EX_NX_LROOT
);

CREATE COLLATION icu_collate_uppercase (
    LOCALE = en_US.UTF8,
    ICU_SHORT_FORM = AN_CU_EX_NX_LROOT
);

CREATE COLLATION icu_collate_ignore_punct (
    LOCALE = en_US.UTF8,
    ICU_SHORT_FORM = AS_CX_EX_NX_LROOT_L3
);

CREATE COLLATION icu_collate_ignore_white_sp (
    LOCALE = en_US.UTF8,
    ICU_SHORT_FORM = AS_CX_EX_NX_LROOT_L3_T0020
);
```

Note: When creating collations, ICU may generate notice and warning messages when attributes are given to modify the *LROOT* collation.

The following `psql` command lists the collations.

```
edb=# \d0
__OUTPUT__

          List of collations
 Schema | Name | Collate | Ctype | ICU
-----+-----+-----+-----+-----
enterprisedb | icu_collate_ignore_punct | en_US.UTF8 | en_US.UTF8 | AS_CX_EX_NX_LROOT_L3
enterprisedb | icu_collate_ignore_white_sp | en_US.UTF8 | en_US.UTF8 | AS_CX_EX_NX_LROOT_L3_T0020
enterprisedb | icu_collate_lowercase | en_US.UTF8 | en_US.UTF8 | AN_CL_EX_NX_LROOT
enterprisedb | icu_collate_uppercase | en_US.UTF8 | en_US.UTF8 | AN_CU_EX_NX_LROOT
(4 rows)
```

The following table is created and populated.

```
CREATE TABLE collate_tbl (
    id          INTEGER,
    c2          VARCHAR(2)
);

INSERT INTO collate_tbl VALUES (1, A);
INSERT INTO collate_tbl VALUES (2, B);
INSERT INTO collate_tbl VALUES (3, C);
```

(continues on next page)

(continued from previous page)

```

INSERT INTO collate_tbl VALUES (4, a);
INSERT INTO collate_tbl VALUES (5, b);
INSERT INTO collate_tbl VALUES (6, c);
INSERT INTO collate_tbl VALUES (7, 1);
INSERT INTO collate_tbl VALUES (8, 2);
INSERT INTO collate_tbl VALUES (9, .B);
INSERT INTO collate_tbl VALUES (10, -B);
INSERT INTO collate_tbl VALUES (11, B);

```

The following query sorts on column c2 using the default collation. Note that variable characters (white space and punctuation marks) with id column values of 9, 10, and 11 are ignored and sort with the letter B.

```
edb=# SELECT * FROM collate_tbl ORDER BY c2;
```

```
__OUTPUT__
```

```

id | c2
---+---
 7 | 1
 8 | 2
 4 | a
 1 | A
 5 | b
 2 | B
11 | B
10 | -B
 9 | .B
 6 | c
 3 | C
(11 rows)

```

The following query sorts on column c2 using collation `icu_collate_lowercase`, which forces the lowercase form of a letter to sort before the uppercase form of the same base letter. Also note that the `AN` attribute forces variable characters to be included in the sort order at the same level when comparing base characters so rows with id values of 9, 10, and 11 appear at the beginning of the sort list before all letters and numbers.

```
edb=# SELECT * FROM collate_tbl ORDER BY c2 COLLATE "icu_collate_lowercase";
```

```
__OUTPUT__
```

```

id | c2
---+---
11 | B
10 | -B
 9 | .B
 7 | 1
 8 | 2
 4 | a
 1 | A
 5 | b
 2 | B
 6 | c
 3 | C
(11 rows)

```

The following query sorts on column c2 using collation `icu_collate_uppercase`, which forces the uppercase form of a letter to sort before the lowercase form of the same base letter.

```

edb=# SELECT * FROM collate_tbl ORDER BY c2 COLLATE "icu_collate_uppercase";
__OUTPUT__
 id | c2
----+----
 11 | B
 10 | -B
  9 | .B
  7 | 1
  8 | 2
  1 | A
  4 | a
  2 | B
  5 | b
  3 | C
  6 | c
(11 rows)

```

The following query sorts on column c2 using collation `icu_collate_ignore_punct`, which causes variable characters to be ignored so rows with id values of 9, 10, and 11 sort with the letter B as that is the character immediately following the ignored variable character.

```

edb=# SELECT * FROM collate_tbl ORDER BY c2 COLLATE
"icu_collate_ignore_punct";
__OUTPUT__
 id | c2
----+----
  7 | 1
  8 | 2
  4 | a
  1 | A
  5 | b
 11 | B
  2 | B
  9 | .B
 10 | -B
  6 | c
  3 | C
(11 rows)

```

The following query sorts on column c2 using collation `icu_collate_ignore_white_sp`. The AS and T0020 attributes of the collation cause variable characters with code points less than or equal to hexadecimal 0020 to be ignored while variable characters with code points greater than hexadecimal 0020 are included in the sort.

The row with id value of 11, which starts with a space character (hexadecimal 0020) sorts with the letter B. The rows with id values of 9 and 10, which start with visible punctuation marks greater than hexadecimal 0020, appear at the beginning of the sort list as these particular variable characters are included in the sort order at the same level when comparing base characters.

```

edb=# SELECT * FROM collate_tbl ORDER BY c2 COLLATE
"icu_collate_ignore_white_sp";
__OUTPUT__
 id | c2
----+----
 10 | -B

```

(continues on next page)

(continued from previous page)

9		.B
7		1
8		2
4		a
1		A
5		b
11		B
2		B
6		c
3		C
(11 rows)		

EDB RESOURCE MANAGER

- EDB Resource Manager* is an EDB Postgres Advanced Server feature that provides the capability to control the usage of operating system resources used by EDB Postgres Advanced Server processes.

This capability allows you to protect the system from processes that may uncontrollably overuse and monopolize certain system resources.

The following are some key points about using EDB Resource Manager.

- The basic component of EDB Resource Manager is a resource group. A *resource group* is a named, global group, available to all databases in an EDB Postgres Advanced Server instance, on which various resource usage limits can be defined. EDB Postgres Advanced Server processes that are assigned as members of a given resource group are then controlled by EDB Resource Manager so that the aggregate resource usage of all processes in the group is kept near the limits defined on the group.
- Data definition language commands are used to create, alter, and drop resource groups. These commands can only be used by a database user with superuser privileges.
- The desired, aggregate consumption level of all processes belonging to a resource group is defined by *resource type parameters*. There are different resource type parameters for the different types of system resources currently supported by EDB Resource Manager.
- Multiple resource groups can be created, each with different settings for its resource type parameters, thus defining different consumption levels for each resource group.
- EDB Resource Manager throttles processes in a resource group to keep resource consumption near the limits defined by the resource type parameters. If there are multiple resource type parameters with defined settings in a resource group, the actual resource consumption may be significantly lower for certain resource types than their defined resource type parameter settings. This is because EDB Resource Manager throttles processes attempting to keep *all resources with defined resource type settings within their defined limits*.
- The definition of available resource groups and their resource type settings are stored in a shared global system catalog. Thus, resource groups can be utilized by all databases in a given EDB Postgres Advanced Server instance.
- The `edb_max_resource_groups` configuration parameter sets the maximum number of resource groups that can be active simultaneously with running processes. The default setting is 16 resource groups. Changes to this parameter take effect on database server restart.
- Use the `SET edb_resource_group TO group_name` command to assign the current process to a specified resource group. Use the `RESET edb_resource_group` command or `SET edb_resource_group TO DEFAULT` to remove the current process from a resource group.
- A default resource group can be assigned to a role using the `ALTER ROLE ... SET` command, or to a database by the `ALTER DATABASE ... SET` command. The entire database server instance can be assigned a default resource group by setting the parameter in the `postgresql.conf` file.

- In order to include resource groups in a backup file of the database server instance, use the `pg_dumpall` backup utility with default settings (That is, don't specify any of the `--globals-only`, `--roles-only`, or `--tablespaces-only` options.)

4.1 Creating and managing resource groups

The data definition language commands described in this section provide for the creation and management of resource groups.

4.1.1 CREATE RESOURCE GROUP

Use the `CREATE RESOURCE GROUP` command to create a new resource group.

```
CREATE RESOURCE GROUP <group_name>;
```

4.1.2 Description

The `CREATE RESOURCE GROUP` command creates a resource group with the specified name. Resource limits can then be defined on the group with the `ALTER RESOURCE GROUP` command. The resource group is accessible from all databases in the EDB Postgres Advanced Server instance.

To use the `CREATE RESOURCE GROUP` command you must have superuser privileges.

4.1.3 Parameters

`group_name`

The name of the resource group.

4.1.4 Example

The following example results in the creation of three resource groups named `resgrp_a`, `resgrp_b`, and `resgrp_c`.

```
edb=# CREATE RESOURCE GROUP resgrp_a;
CREATE RESOURCE GROUP
edb=# CREATE RESOURCE GROUP resgrp_b;
CREATE RESOURCE GROUP
edb=# CREATE RESOURCE GROUP resgrp_c;
CREATE RESOURCE GROUP
```

The following query shows the entries for the resource groups in the `edb_resource_group` catalog.

```
edb=# SELECT * FROM edb_resource_group;
__OUTPUT__
 rgrpname | rgrpcpuratelimt | rgrpdirtyratelimt
-----+-----+-----
 resgrp_a |                0 |                0
 resgrp_b |                0 |                0
 resgrp_c |                0 |                0
(3 rows)
```

4.1.5 ALTER RESOURCE GROUP

Use the `ALTER RESOURCE GROUP` command to change the attributes of an existing resource group. The command syntax comes in three forms.

The first form renames the resource group:

```
ALTER RESOURCE GROUP <group_name> RENAME TO <new_name>;
```

The second form assigns a resource type to the resource group:

```
ALTER RESOURCE GROUP <group_name> SET  
  <resource_type> { TO | = } { <value> | DEFAULT };
```

The third form resets the assignment of a resource type to its default within the group:

```
ALTER RESOURCE GROUP <group_name> RESET <resource_type>;
```

4.1.6 Description

The `ALTER RESOURCE GROUP` command changes certain attributes of an existing resource group.

The first form with the `RENAME TO` clause assigns a new name to an existing resource group.

The second form with the `SET resource_type TO` clause either assigns the specified literal value to a resource type, or resets the resource type when `DEFAULT` is specified. Resetting or setting a resource type to `DEFAULT` means that the resource group has no defined limit on that resource type.

The third form with the `RESET resource_type` clause resets the resource type for the group as described previously.

To use the `ALTER RESOURCE GROUP` command, you must have superuser privileges.

4.1.7 Parameters

group_name

The name of the resource group to be altered.

new_name

The new name to be assigned to the resource group.

resource_type

The resource type parameter specifying the type of resource to which a usage value is to be set.

value | DEFAULT

When **value** is specified, the literal value to be assigned to **resource_type**. When **DEFAULT** is specified, the assignment of **resource_type** is reset for the resource group.

4.1.8 Example

The following are examples of the ALTER RESOURCE GROUP command.

```
edb=# ALTER RESOURCE GROUP resgrp_a RENAME TO newgrp;
ALTER RESOURCE GROUP
edb=# ALTER RESOURCE GROUP resgrp_b SET cpu_rate_limit = .5;
ALTER RESOURCE GROUP
edb=# ALTER RESOURCE GROUP resgrp_b SET dirty_rate_limit = 6144;
ALTER RESOURCE GROUP
edb=# ALTER RESOURCE GROUP resgrp_c RESET cpu_rate_limit;
ALTER RESOURCE GROUP
```

The following query shows the results of the ALTER RESOURCE GROUP commands to the entries in the edb_resource_group catalog.

```
edb=# SELECT * FROM edb_resource_group;
__OUTPUT__
 rgrpname | rgrpcpuratelimit | rgrpdirtyratelimit
-----+-----+-----
 newgrp   | 0                 | 0
 resgrp_b | 0.5               | 6144
 resgrp_c | 0                 | 0
(3 rows)
```

4.1.9 DROP RESOURCE GROUP

Use the DROP RESOURCE GROUP command to remove a resource group.

```
DROP RESOURCE GROUP [ IF EXISTS ] <group_name>;
```

4.1.10 Description

The DROP RESOURCE GROUP command removes a resource group with the specified name.

To use the DROP RESOURCE GROUP command you must have superuser privileges.

4.1.11 Parameters

group_name

The name of the resource group to be removed.

IF EXISTS

Don't throw an error if the resource group does not exist. A notice is issued in this case.

4.1.12 Example

The following example removes resource group newgrp .

```
edb=# DROP RESOURCE GROUP newgrp;
DROP RESOURCE GROUP
```

4.1.13 Assigning a process to a resource group

Use the SET edb_resource_group TO group_name command to assign the current process to a specified resource group as shown by the following.

```
edb=# SET edb_resource_group TO resgrp_b;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----
resgrp_b
(1 row)
```

The resource type settings of the group immediately take effect on the current process. If the command is used to change the resource group assigned to the current process, the resource type settings of the newly assigned group immediately take effect.

Processes can be included by default in a resource group by assigning a default resource group to roles, databases, or an entire database server instance.

A default resource group can be assigned to a role using the ALTER ROLE ... SET command. For more information about the ALTER ROLE command, refer to the PostgreSQL core documentation available at:

<https://www.postgresql.org/docs/current/static/sql-alterrole.html>

A default resource group can be assigned to a database by the ALTER DATABASE ... SET command. For more information about the ALTER DATABASE command, refer to the PostgreSQL core documentation available at:

<https://www.postgresql.org/docs/current/static/sql-alterdatabase.html>

The entire database server instance can be assigned a default resource group by setting the edb_resource_group configuration parameter in the postgresql.conf file as shown by the following.

```
# - EDB Resource Manager -

# edb_max_resource_groups = 16          # 0-65536 (change requires restart)
edb_resource_group = resgrp_b
```

A change to edb_resource_group in the postgresql.conf file requires a configuration file reload before it takes effect on the database server instance.

4.1.14 Removing a process from a resource group

Set `edb_resource_group` to `DEFAULT` or use `RESET edb_resource_group` to remove the current process from a resource group as shown by the following.

```
edb=# SET edb_resource_group TO DEFAULT;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----
(1 row)
```

For removing a default resource group from a role, use the `ALTER ROLE ... RESET` form of the `ALTER ROLE` command.

For removing a default resource group from a database, use the `ALTER DATABASE ... RESET` form of the `ALTER DATABASE` command.

For removing a default resource group from the database server instance, set the `edb_resource_group` configuration parameter to an empty string in the `postgresql.conf` file and reload the configuration file.

4.1.15 Monitoring processes in resource groups

After resource groups have been created, the number of processes actively using these resource groups can be obtained from the view `edb_all_resource_groups`.

The columns in `edb_all_resource_groups` are the following:

- `**group_name.**` Name of the resource group.
- `**active_processes.**` Number of active processes **in** the resource group.
- `**cpu_rate_limit.**` The value of the CPU rate limit resource **type** assigned to the `↪resource group`.
- `**per_process_cpu_rate_limit.**` The CPU rate limit applicable to an individual, `↪active process in the resource group`.
- `**dirty_rate_limit.**` The value of the dirty rate limit resource **type** assigned to `↪the resource group`.
- `**per_process_dirty_rate_limit.**` The dirty rate limit applicable to an individual, `↪ active process in the resource group`.

Note: Columns `per_process_cpu_rate_limit` and `per_process_dirty_rate_limit` don't show the *actual* resource consumption used by the processes, but indicate how *EDB Resource Manager* sets the resource limit for an individual process based upon the number of active processes in the resource group.

The following shows `edb_all_resource_groups` when resource group `resgrp_a` contains no active processes, resource group `resgrp_b` contains two active processes, and resource group `resgrp_c` contains one active process.

```
edb=# SELECT * FROM edb_all_resource_groups ORDER BY group_name;
```

```
__OUTPUT__
```

```
- [ RECORD 1 ]-----+-----
group_name      | resgrp_a
active_processes | 0
cpu_rate_limit  | 0.5
per_process_cpu_rate_limit |
dirty_rate_limit | 12288
per_process_dirty_rate_limit |
- [ RECORD 2 ]-----+-----
group_name      | resgrp_b
active_processes | 2
cpu_rate_limit  | 0.4
per_process_cpu_rate_limit | 0.195694289022895
dirty_rate_limit | 6144
per_process_dirty_rate_limit | 3785.92924684337
- [ RECORD 3 ]-----+-----
group_name      | resgrp_c
active_processes | 1
cpu_rate_limit  | 0.3
per_process_cpu_rate_limit | 0.292342129631091
dirty_rate_limit | 3072
per_process_dirty_rate_limit | 3072
```

The CPU rate limit and dirty rate limit settings that are assigned to these resource groups are as follows.

```
edb=# SELECT * FROM edb_resource_group;
```

```
__OUTPUT__
```

```
  rgrpname  | rgrpcpuratelimit | rgrpdirtyratelimit
-----+-----+-----
resgrp_a   | 0.5              | 12288
resgrp_b   | 0.4              | 6144
resgrp_c   | 0.3              | 3072
(3 rows)
```

In the `edb_all_resource_groups` view, note that the `per_process_cpu_rate_limit` and `per_process_dirty_rate_limit` values are roughly the corresponding CPU rate limit and dirty rate limit divided by the number of active processes.

4.2 CPU usage throttling

CPU usage of a resource group is controlled by setting the `cpu_rate_limit` resource type parameter.

Set the `cpu_rate_limit` parameter to the fraction of CPU time over wall-clock time to which the combined, simultaneous CPU usage of all processes in the group should not exceed. Thus, the value assigned to `cpu_rate_limit` should typically be less than or equal to 1.

On multicore systems, you can apply the `cpu_rate_limit` to more than one CPU core by setting it to greater than 1. For example, if `cpu_rate_limit` is set to 2.0, you use 100% of two CPUs.

The valid range of the `cpu_rate_limit` parameter is 0 to 1.67772e+07. A setting of 0 means no CPU rate limit has been set for the resource group.

When multiplied by 100, the `cpu_rate_limit` can also be interpreted as the CPU usage percentage for a resource group.

EDB Resource Manager utilizes *CPU throttling* to keep the aggregate CPU usage of all processes in the group within the limit specified by the `cpu_rate_limit` parameter. A process in the group may be interrupted and put into sleep mode for a short interval of time to maintain the defined limit. When and how such interruptions occur is defined by a proprietary algorithm used by EDB Resource Manager.

4.2.1 Setting the CPU rate limit for a resource group

The `ALTER RESOURCE GROUP` command with the `SET cpu_rate_limit` clause is used to set the CPU rate limit for a resource group.

In the following example the CPU usage limit is set to 50% for `resgrp_a`, 40% for `resgrp_b` and 30% for `resgrp_c`. This means that the combined CPU usage of all processes assigned to `resgrp_a` is maintained at approximately 50%. Similarly, for all processes in `resgrp_b`, the combined CPU usage is kept to approximately 40%, etc.

```
edb=# ALTER RESOURCE GROUP resgrp_a SET cpu_rate_limit TO .5;
ALTER RESOURCE GROUP
edb=# ALTER RESOURCE GROUP resgrp_b SET cpu_rate_limit TO .4;
ALTER RESOURCE GROUP
edb=# ALTER RESOURCE GROUP resgrp_c SET cpu_rate_limit TO .3;
ALTER RESOURCE GROUP
```

The following query shows the settings of `cpu_rate_limit` in the catalog.

```
edb=# SELECT rgrpname, rgrpcpuratelimit FROM edb_resource_group;
__OUTPUT__
 rgrpname | rgrpcpuratelimit
-----+-----
 resgrp_a | 0.5
 resgrp_b | 0.4
 resgrp_c | 0.3
(3 rows)
```

Changing the `cpu_rate_limit` of a resource group not only affects new processes that are assigned to the group, but any currently running processes that are members of the group are immediately affected by the change. That is, if the `cpu_rate_limit` is changed from .5 to .3, currently running processes in the group would be throttled downward so that the aggregate group CPU usage would be near 30% instead of 50%.

To illustrate the effect of setting the CPU rate limit for resource groups, the following examples use a CPU-intensive calculation of 20000 factorial (multiplication of 20000 * 19999 * 19998, etc.) performed by the query `SELECT 20000!`; run in the `psql` command line utility.

The resource groups with the CPU rate limit settings shown in the previous query are used in these examples.

4.2.2 Example: Single process in a single group

The following shows that the current process is set to use resource group `resgrp_b`. The factorial calculation is then started.

```
edb=# SET edb_resource_group TO resgrp_b;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----
resgrp_b
(1 row)
edb=# SELECT 20000!;
```

In a second session, the Linux `top` command is used to display the CPU usage as shown under the `%CPU` column. The following is a snapshot at an arbitrary point in time as the `top` command output periodically changes.

```
$ top
top - 16:37:03 up 4:15, 7 users, load average: 0.49, 0.20, 0.38
Tasks: 202 total, 1 running, 201 sleeping, 0 stopped, 0 zombie
Cpu(s): 42.7%us, 2.3%sy, 0.0%ni, 55.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0
Mem: 1025624k total, 791160k used, 234464k free, 23400k buffers
Swap: 103420k total, 13404k used, 90016k free, 373504k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  COMMAND
28915 enterpri  20   0 195m 5900 4212 S 39.9   0.6   3:36.98 edb-postgres
1033  root      20   0 171m  77m 2960 S  1.0   7.8   3:43.96 Xorg
3040  user      20   0 278m  22m 14m  S  1.0   2.2   3:41.72 knotify4
.
.
.
```

The `psql` session performing the factorial calculation is shown by the row where `edb-postgres` appears under the `COMMAND` column. The CPU usage of the session shown under the `%CPU` column shows 39.9, which is close to the 40% CPU limit set for resource group `resgrp_b`.

By contrast, if the `psql` session is removed from the resource group and the factorial calculation is performed again, the CPU usage is much higher.

```
edb=# SET edb_resource_group TO DEFAULT;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----

(1 row)

edb=# SELECT 20000!;
```

Under the `%CPU` column for `edb-postgres`, the CPU usage is now 93.6, which is significantly higher than the 39.9 when the process was part of the resource group.

```
$ top
top - 16:43:03 up 4:21, 7 users, load average: 0.66, 0.33, 0.37
Tasks: 202 total, 5 running, 197 sleeping, 0 stopped, 0 zombie
Cpu(s): 96.7%us, 3.3%sy, 0.0%ni, 0.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0
Mem: 1025624k total, 791228k used, 234396k free, 23560k buffers
Swap: 103420k total, 13404k used, 90016k free, 373508k cached

  PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
28915 enterpri 20 0 195m 5900 4212 R 93.6 0.6 5:01.56 edb-postgres
1033 root 20 0 171m 77m 2960 S 1.0 7.8 3:48.15 Xorg
2907 user 20 0 98.7m 11m 9100 S 0.3 1.2 0:46.51 vmware-user-lo
.
.
.
```

4.2.3 Example: Multiple processes in a single group

As stated previously, the CPU rate limit applies to the aggregate of all processes in the resource group. This concept is illustrated in the following example.

The factorial calculation is performed simultaneously in two separate `psql` sessions, each of which has been added to resource group `resgrp_b` that has `cpu_rate_limit` set to `.4` (CPU usage of 40%).

Session1:

```
edb=# SET edb_resource_group TO resgrp_b;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----
resgrp_b
(1 row)

edb=# SELECT 20000!;
```

Session2:

```
edb=# SET edb_resource_group TO resgrp_b;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----
resgrp_b
(1 row)

edb=# SELECT 20000!;
```

A third session monitors the CPU usage.

```
$ top
top - 16:53:03 up 4:31, 7 users, load average: 0.31, 0.19, 0.27
```

(continues on next page)

(continued from previous page)

```
Tasks: 202 total, 1 running, 201 sleeping, 0 stopped, 0 zombie
Cpu(s): 41.2%us, 3.0%sy, 0.0%ni, 55.8%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0
Mem: 1025624k total, 792020k used, 233604k free, 23844k buffers
Swap: 103420k total, 13404k used, 90016k free, 373508k cached
```

```

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
29857 enterpri 20 0 195m 4708 3312 S 19.9 0.5 0:57.35 edb-postgres
28915 enterpri 20 0 195m 5900 4212 S 19.6 0.6 5:35.49 edb-postgres
3040 user 20 0 278m 22m 14m S 1.0 2.2 3:54.99 knotify4
1033 root 20 0 171m 78m 2960 S 0.3 7.8 3:55.71 Xorg
.
.
.
```

There are now two processes named `edb-postgres` with %CPU values of 19.9 and 19.6, whose sum is close to the 40% CPU usage set for resource group `resgrp_b`.

The following command sequence displays the sum of all `edb-postgres` processes sampled over half second time intervals. This shows how the total CPU usage of the processes in the resource group changes over time as EDB Resource Manager throttles the processes to keep the total resource group CPU usage near 40%.

```

$ while [[ 1 -eq 1 ]]; do top -d0.5 -b -n2 | grep edb-postgres | awk { SUM
+= $9 } END { print SUM / 2 }; done
37.2
39.1
38.9
38.3
44.7
39.2
42.5
39.1
39.2
39.2
41
42.85
46.1
.
.
.
```

4.2.4 Example: Multiple processes in multiple groups

In this example, two additional `psql` sessions are used along with the previous two sessions. The third and fourth sessions perform the same factorial calculation within resource group `resgrp_c` with a `cpu_rate_limit` of .3 (30% CPU usage).

Session3:

```

edb=# SET edb_resource_group TO resgrp_c;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
```

(continues on next page)

(continued from previous page)

```
edb_resource_group
-----
 resgrp_c
(1 row)

edb=# SELECT 20000!;
```

Session4:

```
edb=# SET edb_resource_group TO resgrp_c;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----
 resgrp_c
(1 row)
edb=# SELECT 20000!;
```

The top command displays the following output.

```
$ top
top - 17:45:09 up 5:23, 8 users, load average: 0.47, 0.17, 0.26
Tasks: 203 total, 4 running, 199 sleeping, 0 stopped, 0 zombie
Cpu(s): 70.2%us, 0.0%sy, 0.0%ni, 29.8%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 1025624k total, 806140k used, 219484k free, 25296k buffers
Swap: 103420k total, 13404k used, 90016k free, 374092k cached

  PID   USER     PR  NI  VIRT  RES   SHR   S  %CPU  %MEM  TIME+  COMMAND
29857  enterpri 20   0   195m 4820 3324  S  19.9  0.5   4:25.02 edb-postgres
28915  enterpri 20   0   195m 5900 4212  R  19.6  0.6   9:07.50 edb-postgres
29023  enterpri 20   0   195m 4744 3248  R  16.3  0.5   4:01.73 edb-postgres
11019  enterpri 20   0   195m 4120 2764  R  15.3  0.4   0:04.92 edb-postgres
 2907   user      20   0   98.7m 12m  9112  S   1.3  1.2   0:56.54 vmware-user-lo
 3040   user      20   0   278m  22m  14m  S   1.3  2.2   4:38.73 knotify4
```

The two resource groups in use have CPU usage limits of 40% and 30%. The sum of the %CPU column for the first two edb-postgres processes is 39.5 (approximately 40%, which is the limit for resgrp_b) and the sum of the %CPU column for the third and fourth edb-postgres processes is 31.6 (approximately 30%, which is the limit for resgrp_c).

The sum of the CPU usage limits of the two resource groups to which these processes belong is 70%. The following output shows that the sum of the four processes borders around 70%.

```
$ while [[ 1 -eq 1 ]]; do top -d0.5 -b -n2 | grep edb-postgres | awk { SUM
+= $9 } END { print SUM / 2 }; done
61.8
76.4
72.6
69.55
64.55
79.95
68.55
```

(continues on next page)

(continued from previous page)

```

71.25
74.85
62
74.85
76.9
72.4
65.9
74.9
68.25

```

By contrast, if three sessions are processing where two sessions remain in `resgrp_b`, but the third session does not belong to any resource group, the `top` command shows the following output.

```

$ top
top - 17:24:55 up 5:03, 7 users, load average: 1.00, 0.41, 0.38
Tasks: 199 total, 3 running, 196 sleeping, 0 stopped, 0 zombie
Cpu(s): 99.7%us, 0.3%sy, 0.0%ni, 0.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0
Mem: 1025624k total, 797692k used, 227932k free, 24724k buffers
Swap: 103420k total, 13404k used, 90016k free, 374068k cached

  PID  USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM  TIME+  COMMAND
29023  enterpri  20   0   195m  4744 3248 R   58.6   0.5   2:53.75 edb-postgres
28915  enterpri  20   0   195m  5900 4212 S   18.9   0.6   7:58.45 edb-postgres
29857  enterpri  20   0   195m  4820 3324 S   18.9   0.5   3:14.85 edb-postgres
 1033  root      20   0   174m   81m 2960 S    1.7   8.2   4:26.50 Xorg
 3040  user      20   0   278m   22m 14m S    1.0   2.2   4:21.20 knotify4

```

The second and third `edb-postgres` processes belonging to the resource group where the CPU usage is limited to 40%, have a total CPU usage of 37.8. However, the first `edb-postgres` process has a 58.6% CPU usage as it is not within a resource group, and basically utilizes the remaining, available CPU resources on the system.

Likewise, the following output shows the sum of all three sessions is around 95% since one of the sessions has no set limit on its CPU usage.

```

$ while [[ 1 -eq 1 ]]; do top -d0.5 -b -n2 | grep edb-postgres | awk { SUM
+= $9 } END { print SUM / 2 }; done
96
90.35
92.55
96.4
94.1
90.7
95.7
95.45
93.65
87.95
96.75
94.25
95.45
97.35
92.9
96.05
96.25

```

(continues on next page)

(continued from previous page)

94.95

.
.
.

4.3 Dirty buffer throttling

Writing to shared buffers is controlled by setting the `dirty_rate_limit` resource type parameter.

Set the `dirty_rate_limit` parameter to the number of kilobytes per second for the combined rate at which all the processes in the group should write to or “dirty” the shared buffers. An example setting would be 3072 kilobytes per seconds.

The valid range of the `dirty_rate_limit` parameter is 0 to 1.67772e+07 . A setting of 0 means no dirty rate limit has been set for the resource group.

EDB Resource Manager utilizes *dirty buffer throttling* to keep the aggregate, shared buffer writing rate of all processes in the group near the limit specified by the `dirty_rate_limit` parameter. A process in the group may be interrupted and put into sleep mode for a short interval of time to maintain the defined limit. When and how such interruptions occur is defined by a proprietary algorithm used by EDB Resource Manager.

4.3.1 Setting the dirty rate limit for a resource group

The `ALTER RESOURCE GROUP` command with the `SET dirty_rate_limit` clause is used to set the dirty rate limit for a resource group.

In the following example the dirty rate limit is set to 12288 kilobytes per second for `resgrp_a` , 6144 kilobytes per second for `resgrp_b` and 3072 kilobytes per second for `resgrp_c` . This means that the combined writing rate to the shared buffer of all processes assigned to `resgrp_a` is maintained at approximately 12288 kilobytes per second. Similarly, for all processes in `resgrp_b` , the combined writing rate to the shared buffer is kept to approximately 6144 kilobytes per second, etc.

```
edb=# ALTER RESOURCE GROUP resgrp_a SET dirty_rate_limit TO 12288;
ALTER RESOURCE GROUP
edb=# ALTER RESOURCE GROUP resgrp_b SET dirty_rate_limit TO 6144;
ALTER RESOURCE GROUP
edb=# ALTER RESOURCE GROUP resgrp_c SET dirty_rate_limit TO 3072;
ALTER RESOURCE GROUP
```

The following query shows the settings of `dirty_rate_limit` in the catalog.

```
edb=# SELECT rgrpname, rgrpdirtyratelimit FROM edb_resource_group;
__OUTPUT__
 rgrpname | rgrpdirtyratelimit
-----+-----
 resgrp_a |          12288
 resgrp_b |           6144
 resgrp_c |           3072
(3 rows)
```

Changing the `dirty_rate_limit` of a resource group not only affects new processes that are assigned to the group, but any currently running processes that are members of the group are immediately affected by the change. That is, if

the `dirty_rate_limit` is changed from 12288 to 3072, currently running processes in the group would be throttled downward so that the aggregate group dirty rate would be near 3072 kilobytes per second instead of 12288 kilobytes per second.

To illustrate the effect of setting the dirty rate limit for resource groups, the following examples use the following table for intensive I/O operations.

```
CREATE TABLE t1 (c1 INTEGER, c2 CHARACTER(500)) WITH (FILLFACTOR = 10);
```

The `FILLFACTOR = 10` clause results in `INSERT` commands packing rows up to only 10% per page. This results in a larger sampling of dirty shared blocks for the purpose of these examples.

The `pg_stat_statements` module is used to display the number of shared buffer blocks that are dirtied by a SQL command and the amount of time the command took to execute. This provides the information to calculate the actual kilobytes per second writing rate for the SQL command, and thus compare it to the dirty rate limit set for a resource group.

In order to use the `pg_stat_statements` module, perform the following steps.

Step1: In the `postgresql.conf` file, add `$libdir/pg_stat_statements` to the `shared_preload_libraries` configuration parameter as shown by the following.

```
shared_preload_libraries = $libdir/dbms_pipe,$libdir/edb_gen,$libdir/pg_stat_statements
```

Step2: Restart the database server.

Step3: Use the `CREATE EXTENSION` command to complete the creation of the `pg_stat_statements` module.

```
edb=# CREATE EXTENSION pg_stat_statements SCHEMA public;
CREATE EXTENSION
```

The `pg_stat_statements_reset()` function is used to clear out the `pg_stat_statements` view for clarity of each example.

The resource groups with the dirty rate limit settings shown in the previous query are used in these examples.

4.3.2 Example: Single process in a single group

The following sequence of commands shows the creation of table `t1`. The current process is set to use resource group `resgrp_b`. The `pg_stat_statements` view is cleared out by running the `pg_stat_statements_reset()` function.

Finally, the `INSERT` command generates a series of integers from 1 to 10,000 to populate the table, and dirty approximately 10,000 blocks.

```
edb=# CREATE TABLE t1 (c1 INTEGER, c2 CHARACTER(500)) WITH (FILLFACTOR = 10);
CREATE TABLE
edb=# SET edb_resource_group TO resgrp_b;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----
resgrp_b
(1 row)
```

```

edb=# SELECT pg_stat_statements_reset();
__OUTPUT__
pg_stat_statements_reset
-----

(1 row)

edb=# INSERT INTO t1 VALUES (generate_series (1,10000), aaa);
INSERT 0 10000

```

The following shows the results from the INSERT command.

```

edb=# SELECT query, rows, total_time, shared_blks_dirtied FROM
pg_stat_statements;
__OUTPUT__
- [ RECORD 1 ]-----+-----
query          | INSERT INTO t1 VALUES (generate_series (?,?), ?);
rows           | 10000
total_time     | 13496.184
shared_blks_dirtied | 10003

```

The actual dirty rate is calculated as follows.

- The number of blocks dirtied per millisecond (ms) is 10003 blocks / 13496.184 ms, which yields *0.74117247 blocks per millisecond*.
- Multiply the result by 1000 to give the number of shared blocks dirtied per second (1 second = 1000 ms), which yields *741.17247 blocks per second*.
- Multiply the result by 8.192 to give the number of kilobytes dirtied per second (1 block = 8.192 kilobytes), which yields approximately *6072 kilobytes per second*.

Note that the actual dirty rate of 6072 kilobytes per second is close to the dirty rate limit for the resource group, which is 6144 kilobytes per second.

By contrast, if the steps are repeated again without the process belonging to any resource group, the dirty buffer rate is much higher.

```

edb=# CREATE TABLE t1 (c1 INTEGER, c2 CHARACTER(500)) WITH (FILLFACTOR = 10);
CREATE TABLE
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----

(1 row)

```

```

edb=# SELECT pg_stat_statements_reset();
__OUTPUT__
pg_stat_statements_reset
-----

(1 row)

edb=# INSERT INTO t1 VALUES (generate_series (1,10000), aaa);
INSERT 0 10000

```

The following shows the results from the INSERT command without the usage of a resource group.

```
edb=# SELECT query, rows, total_time, shared_blks_dirtied FROM
pg_stat_statements;
__OUTPUT__
- [ RECORD 1 ]-----+-----
query          | INSERT INTO t1 VALUES (generate_series (?,?), ?);
rows           | 10000
total_time     | 2432.165
shared_blks_dirtied | 10003
```

First, note the total time was only 2432.165 milliseconds as compared to 13496.184 milliseconds when a resource group with a dirty rate limit set to 6144 kilobytes per second was used.

The actual dirty rate without the use of a resource group is calculated as follows.

- The number of blocks dirtied per millisecond (ms) is 10003 blocks / 2432.165 ms, which yields *4.112797 blocks per millisecond*.
- Multiply the result by 1000 to give the number of shared blocks dirtied per second (1 second = 1000 ms), which yields *4112.797 blocks per second*.
- Multiply the result by 8.192 to give the number of kilobytes dirtied per second (1 block = 8.192 kilobytes), which yields approximately *33692 kilobytes per second*.

Note that the actual dirty rate of 33692 kilobytes per second is significantly higher than when the resource group with a dirty rate limit of 6144 kilobytes per second was used.

4.3.3 Example: Multiple processes in a single group

As stated previously, the dirty rate limit applies to the aggregate of all processes in the resource group. This concept is illustrated in the following example.

For this example the inserts are performed simultaneously on two different tables in two separate psql sessions, each of which has been added to resource group resgrp_b that has a dirty_rate_limit set to 6144 kilobytes per second.

Session1:

```
edb=# CREATE TABLE t1 (c1 INTEGER, c2 CHARACTER(500)) WITH (FILLFACTOR = 10);
CREATE TABLE
edb=# SET edb_resource_group TO resgrp_b;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----
resgrp_b
(1 row)

edb=# INSERT INTO t1 VALUES (generate_series (1,10000), aaa);
INSERT 0 10000
```

Session2:

```
edb=# CREATE TABLE t2 (c1 INTEGER, c2 CHARACTER(500)) WITH (FILLFACTOR = 10);
CREATE TABLE
```

(continues on next page)

(continued from previous page)

```

edb=# SET edb_resource_group TO resgrp_b;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
- -----
resgrp_b
(1 row)

```

```

edb=# SELECT pg_stat_statements_reset();
__OUTPUT__
pg_stat_statements_reset
- -----
(1 row)

edb=# INSERT INTO t2 VALUES (generate_series (1,10000), aaa);
INSERT 0 10000

```

Note: The *INSERT* commands in session 1 and session 2 were started after the *SELECT pg_stat_statements_reset()* command in session 2 was run.

The following shows the results from the *INSERT* commands in the two sessions. RECORD 3 shows the results from session 1. RECORD 2 shows the results from session 2.

```

edb=# SELECT query, rows, total_time, shared_blks_dirtied FROM
pg_stat_statements;
__OUTPUT__
- [ RECORD 1 ]-----+-----
query          | SELECT pg_stat_statements_reset();
rows           | 1
total_time     | 0.43
shared_blks_dirtied | 0
- [ RECORD 2 ]-----+-----
query          | INSERT INTO t2 VALUES (generate_series (?,?), ?);
rows           | 10000
total_time     | 30591.551
shared_blks_dirtied | 10003
- [ RECORD 3 ]-----+-----
query          | INSERT INTO t1 VALUES (generate_series (?,?), ?);
rows           | 10000
total_time     | 33215.334
shared_blks_dirtied | 10003

```

First, note the total time was 33215.334 milliseconds for session 1 and 30591.551 milliseconds for session 2. When only one session was active in the same resource group as shown in the first example, the time was 13496.184 milliseconds. Thus more active processes in the resource group result in a slower dirty rate for each active process in the group. This is shown in the following calculations.

The actual dirty rate for session 1 is calculated as follows.

- The number of blocks dirtied per millisecond (ms) is 10003 blocks / 33215.334 ms, which yields *0.30115609 blocks per millisecond*.

- Multiply the result by 1000 to give the number of shared blocks dirtied per second (1 second = 1000 ms), which yields *301.15609 blocks per second*.
- Multiply the result by 8.192 to give the number of kilobytes dirtied per second (1 block = 8.192 kilobytes), which yields approximately *2467 kilobytes per second*.

The actual dirty rate for session 2 is calculated as follows.

- The number of blocks dirtied per millisecond (ms) is 10003 blocks / 30591.551 ms, which yields *0.32698571 blocks per millisecond*.
- Multiply the result by 1000 to give the number of shared blocks dirtied per second (1 second = 1000 ms), which yields *326.98571 blocks per second*.
- Multiply the result by 8.192 to give the number of kilobytes dirtied per second (1 block = 8.192 kilobytes), which yields approximately *2679 kilobytes per second*.

The combined dirty rate from session 1 (2467 kilobytes per second) and from session 2 (2679 kilobytes per second) yields 5146 kilobytes per second, which is below the set dirty rate limit of the resource group (6144 kilobytes per seconds).

4.3.4 Example: Multiple processes in multiple groups

In this example, two additional psql sessions are used along with the previous two sessions. The third and fourth sessions perform the same INSERT command in resource group resgrp_c with a dirty_rate_limit of 3072 kilobytes per second.

Sessions 1 and 2 are repeated as illustrated in the prior example using resource group resgrp_b with a dirty_rate_limit of 6144 kilobytes per second.

Session3:

```
edb=# CREATE TABLE t3 (c1 INTEGER, c2 CHARACTER(500)) WITH (FILLFACTOR = 10);
CREATE TABLE
edb=# SET edb_resource_group TO resgrp_c;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----

resgrp_c
(1 row)

edb=# INSERT INTO t3 VALUES (generate_series (1,10000), aaa);
INSERT 0 10000
```

Session4:

```
edb=# CREATE TABLE t4 (c1 INTEGER, c2 CHARACTER(500)) WITH (FILLFACTOR = 10);
CREATE TABLE
edb=# SET edb_resource_group TO resgrp_c;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----
```

(continues on next page)

(continued from previous page)

```
resgrp_c
(1 row)
```

```
edb=# SELECT pg_stat_statements_reset();
__OUTPUT__
pg_stat_statements_reset
-----

(1 row)

edb=# INSERT INTO t4 VALUES (generate_series (1,10000), aaa);
INSERT 0 10000
```

Note: The *INSERT* commands in all four sessions were started after the *SELECT pg_stat_statements_reset()* command in session 4 was run.

The following shows the results from the *INSERT* commands in the four sessions.

RECORD 3 shows the results from session 1. RECORD 2 shows the results from session 2.

RECORD 4 shows the results from session 3. RECORD 5 shows the results from session 4.

```
edb=# SELECT query, rows, total_time, shared_blks_dirtied FROM
pg_stat_statements;
__OUTPUT__
- [ RECORD 1 ]-----+-----
query           | SELECT pg_stat_statements_reset();
rows            | 1
total_time      | 0.467
shared_blks_dirtied | 0
- [ RECORD 2 ]-----+-----
query           | INSERT INTO t2 VALUES (generate_series (?,?), ?);
rows            | 10000
total_time      | 31343.458
shared_blks_dirtied | 10003
- [ RECORD 3 ]-----+-----
query           | INSERT INTO t1 VALUES (generate_series (?,?), ?);
rows            | 10000
total_time      | 28407.435
shared_blks_dirtied | 10003
- [ RECORD 4 ]-----+-----
query           | INSERT INTO t3 VALUES (generate_series (?,?), ?);
rows            | 10000
total_time      | 52727.846
shared_blks_dirtied | 10003
- [ RECORD 5 ]-----+-----
query           | INSERT INTO t4 VALUES (generate_series (?,?), ?);
rows            | 10000
total_time      | 56063.697
shared_blks_dirtied | 10003
```

First note that the times of session 1 (28407.435) and session 2 (31343.458) are close to each other as they are both in

the same resource group with `dirty_rate_limit` set to 6144, as compared to the times of session 3 (52727.846) and session 4 (56063.697), which are in the resource group with `dirty_rate_limit` set to 3072. The latter group has a slower dirty rate limit so the expected processing time is longer as is the case for sessions 3 and 4.

The actual dirty rate for session 1 is calculated as follows.

- The number of blocks dirtied per millisecond (ms) is 10003 blocks / 28407.435 ms, which yields *0.35212612 blocks per millisecond*.
- Multiply the result by 1000 to give the number of shared blocks dirtied per second (1 second = 1000 ms), which yields *352.12612 blocks per second*.
- Multiply the result by 8.192 to give the number of kilobytes dirtied per second (1 block = 8.192 kilobytes), which yields approximately *2885 kilobytes per second*.

The actual dirty rate for session 2 is calculated as follows.

- The number of blocks dirtied per millisecond (ms) is 10003 blocks / 31343.458 ms, which yields *0.31914156 blocks per millisecond*.
- Multiply the result by 1000 to give the number of shared blocks dirtied per second (1 second = 1000 ms), which yields *319.14156 blocks per second*.
- Multiply the result by 8.192 to give the number of kilobytes dirtied per second (1 block = 8.192 kilobytes), which yields approximately *2614 kilobytes per second*.

The combined dirty rate from session 1 (2885 kilobytes per second) and from session 2 (2614 kilobytes per second) yields 5499 kilobytes per second, which is near the set dirty rate limit of the resource group (6144 kilobytes per seconds).

The actual dirty rate for session 3 is calculated as follows.

- The number of blocks dirtied per millisecond (ms) is 10003 blocks / 52727.846 ms, which yields *0.18971001 blocks per millisecond*.
- Multiply the result by 1000 to give the number of shared blocks dirtied per second (1 second = 1000 ms), which yields *189.71001 blocks per second*.
- Multiply the result by 8.192 to give the number of kilobytes dirtied per second (1 block = 8.192 kilobytes), which yields approximately *1554 kilobytes per second*.

The actual dirty rate for session 4 is calculated as follows.

- The number of blocks dirtied per millisecond (ms) is 10003 blocks / 56063.697 ms, which yields *0.17842205 blocks per millisecond*.
- Multiply the result by 1000 to give the number of shared blocks dirtied per second (1 second = 1000 ms), which yields *178.42205 blocks per second*.
- Multiply the result by 8.192 to give the number of kilobytes dirtied per second (1 block = 8.192 kilobytes), which yields approximately *1462 kilobytes per second*.

The combined dirty rate from session 3 (1554 kilobytes per second) and from session 4 (1462 kilobytes per second) yields 3016 kilobytes per second, which is near the set dirty rate limit of the resource group (3072 kilobytes per seconds).

Thus, this demonstrates how EDB Resource Manager keeps the aggregate dirty rate of the active processes in its groups close to the dirty rate limit set for each group.

4.4 System catalogs

This section describes the system catalogs that store the resource group information used by EDB Resource Manager.

4.4.1 edb_all_resource_groups

The following table lists the information available in the `edb_all_resource_groups` catalog:

Column	Type	Description
<i>group_name</i>	<i>name</i>	The name of the resource group.
<i>active_processes</i>	<i>integer</i>	Number of currently active processes in the resource group.
<i>cpu_rate_limit</i>	<i>float8</i>	Maximum CPU rate limit for the resource group. <i>0</i> means no limit.
<i>per_process_cpu_rate_limit</i>	<i>float8</i>	Maximum CPU rate limit per currently active process in the resource group.
<i>dirty_rate_limit</i>	<i>float8</i>	Maximum dirty rate limit for a resource group. <i>0</i> means no limit.
<i>per_process_dirty_rate_limit</i>	<i>float8</i>	Maximum dirty rate limit per currently active process in the resource group.

4.4.2 edb_resource_group

The following table lists the information available in the `edb_resource_group` catalog:

Column	Type	Description
<i>rgrpname</i>	<i>name</i>	The name of the resource group.
<i>rgrpcpuratelimit</i>	<i>float8</i>	Maximum CPU rate limit for a resource group. <i>0</i> means no limit.
<i>rgrpdirtyratelimit</i>	<i>float8</i>	Maximum dirty rate limit for a resource group. <i>0</i> means no limit.

LIBPQ C LIBRARY

libpq is the C application programmer's interface to EDB Postgres Advanced Server. libpq is a set of library functions that allow client programs to pass queries to the EDB Postgres Advanced Server and to receive the results of these queries.

libpq is also the underlying engine for several other EDB application interfaces including those written for C++, Perl, Python, Tcl and ECPG. So some aspects of libpq's behavior are important to the user if one of those packages is used.

Client programs that use libpq must include the header file `libpq-fe.h` and must link with the libpq library.

5.1 Using libpq with EDB SPL

The EDB SPL language can be used with the libpq interface library, providing support for:

- Procedures, functions, packages
- Prepared statements
- REFCURSORs
- Static cursors
- structs and typedefs
- Arrays
- DML and DDL operations
- IN /OUT /IN OUT parameters

5.2 REFCURSOR support

In earlier releases, EDB Postgres Advanced Server provided support for REFCURSORs through the following libpq functions; these functions should now be considered deprecated:

- `PQCursorResult()`
- `PQgetCursorResult()`
- `PQnCursor()`

You may now use `PQexec()` and `PQgetvalue()` to retrieve a REFCURSOR returned by an SPL (or PL/pgSQL) function. A REFCURSOR is returned in the form of a null-terminated string indicating the name of the cursor. Once you have the name of the cursor, you can execute one or more `FETCH` statements to retrieve the values exposed through the cursor.

The samples that follow don't include the error-handling code required in a real-world client application.

5.3 Returning a single REFCURSOR

The following example shows an SPL function that returns a value of type REFCURSOR :

```
CREATE OR REPLACE FUNCTION getEmployees(p_deptno NUMERIC)
RETURN REFCURSOR AS
    result REFCURSOR;
BEGIN
    OPEN result FOR SELECT * FROM emp WHERE deptno = p_deptno;

    RETURN result;
END;
```

This function expects a single parameter, `p_deptno`, and returns a REFCURSOR that holds the result set for the SELECT query shown in the OPEN statement. The OPEN statement executes the query and stores the result set in a cursor. The server constructs a name for that cursor and stores the name in a variable (named `result`). The function then returns the name of the cursor to the caller.

To call this function from a C client using libpq, you can use `PQexec()` and `PQgetvalue()` :

```
# include <stdio.h>
# include <stdlib.h>
# include "libpq-fe.h"

static void fetchAllRows(PGconn *conn,
                        const char *cursorName,
                        const char *description);
static void fail(PGconn *conn, const char *msg);

int
main(int argc, char *argv[])
{
    PGconn    *conn = PQconnectdb(argv[1]);
    PGresult  *result;

    if (PQstatus(conn) != CONNECTION_OK)
        fail(conn, PQerrorMessage(conn));

    result = PQexec(conn, "BEGIN TRANSACTION");

    if (PQresultStatus(result) != PGRES_COMMAND_OK)
        fail(conn, PQerrorMessage(conn));

    PQclear(result);

    result = PQexec(conn, "SELECT * FROM getEmployees(10)");

    if (PQresultStatus(result) != PGRES_TUPLES_OK)
        fail(conn, PQerrorMessage(conn));

    fetchAllRows(conn, PQgetvalue(result, 0, 0), "employees");

    PQclear(result);
```

(continues on next page)

(continued from previous page)

```

PQexec(conn, "COMMIT");

PQfinish(conn);

exit(0);
}

static void
fetchAllRows(PGconn *conn,
             const char *cursorName,
             const char *description)
{
    size_t  commandLength = strlen("FETCH ALL FROM ") +
                           strlen(cursorName) + 3;

    char    *commandText = malloc(commandLength);
    PGresult *result;
    int      row;

    sprintf(commandText, "FETCH ALL FROM \"%s\"", cursorName);

    result = PQexec(conn, commandText);

    if (PQresultStatus(result) != PGRES_TUPLES_OK)
        fail(conn, PQerrorMessage(conn));

    printf("-- %s --\n", description);

    for (row = 0; row < PQntuples(result); row++)
    {
        const char *delimiter = "\t";
        int         col;

        for (col = 0; col < PQnfields(result); col++)
        {
            printf("%s%s", delimiter, PQgetvalue(result, row, col));
            delimiter = ",";
        }

        printf("\n");
    }

    PQclear(result);
    free(commandText);
}

static void
fail(PGconn *conn, const char *msg)
{
    fprintf(stderr, "%s\n", msg);
}

```

(continues on next page)

(continued from previous page)

```

if (conn != NULL)
    PQfinish(conn);

exit(-1);
}

```

The code sample contains a line of code that calls the `getEmployees()` function, and returns a result set that contains all of the employees in department 10 :

```
result = PQexec(conn, "SELECT * FROM getEmployees(10)");
```

The `PQexec()` function returns a result set handle to the C program. The result set contains exactly one value; that value is the name of the cursor as returned by `getEmployees()` .

Once you have the name of the cursor, you can use the SQL `FETCH` statement to retrieve the rows in that cursor. The function `fetchAllRows()` builds a `FETCH ALL` statement, executes that statement, and then prints the result set of the `FETCH ALL` statement.

The output of this program is shown below:

```

- - employees - -
7782,CLARK,MANAGER,7839,09-JUN-81 00:00:00,2450.00,,10
7839,KING,PRESIDENT,,17-NOV-81 00:00:00,5000.00,,10
7934,MILLER,CLERK,7782,23-JAN-82 00:00:00,1300.00,,10

```

5.3.1 Returning multiple REFCURSORs

The next example returns two REFCURSORs :

- The first REFCURSOR contains the name of a cursor (`employees`) that contains all employees who work in a department within the range specified by the caller.
- The second REFCURSOR contains the name of a cursor (`departments`) that contains all of the departments in the range specified by the caller.

In this example, instead of returning a single REFCURSOR , the function returns a `SETOF REFCURSOR` (which means 0 or more REFCURSORs). One other important difference is that the `libpq` program should not expect a single REFCURSOR in the result set, but should expect two rows, each of which contains a single value (the first row contains the name of the `employees` cursor, and the second row contains the name of the `departments` cursor).

```

CREATE OR REPLACE FUNCTION getEmpsAndDepts(p_min NUMERIC,
                                           p_max NUMERIC)
RETURN SETOF REFCURSOR AS
    employees REFCURSOR;
    departments REFCURSOR;
BEGIN
    OPEN employees FOR
        SELECT * FROM emp WHERE deptno BETWEEN p_min AND p_max;
    RETURN NEXT employees;

    OPEN departments FOR
        SELECT * FROM dept WHERE deptno BETWEEN p_min AND p_max;
    RETURN NEXT departments;
END;

```

As in the previous example, you can use PQexec() and PQgetvalue() to call the SPL function:

```
# include <stdio.h>
# include <stdlib.h>
# include "libpq-fe.h"

static void fetchAllRows(PGconn *conn,
                        const char *cursorName,
                        const char *description);
static void fail(PGconn *conn, const char *msg);

int
main(int argc, char *argv[])
{
    PGconn *conn = PQconnectdb(argv[1]);
    PGresult *result;

    if (PQstatus(conn) != CONNECTION_OK)
        fail(conn, PQerrorMessage(conn));

    result = PQexec(conn, "BEGIN TRANSACTION");

    if (PQresultStatus(result) != PGRES_COMMAND_OK)
        fail(conn, PQerrorMessage(conn));

    PQclear(result);

    result = PQexec(conn, "SELECT * FROM getEmpsAndDepts(20, 30)");

    if (PQresultStatus(result) != PGRES_TUPLES_OK)
        fail(conn, PQerrorMessage(conn));

    fetchAllRows(conn, PQgetvalue(result, 0, 0), "employees");
    fetchAllRows(conn, PQgetvalue(result, 1, 0), "departments");

    PQclear(result);

    PQexec(conn, "COMMIT");

    PQfinish(conn);

    exit(0);
}

static void
fetchAllRows(PGconn *conn,
             const char *cursorName,
             const char *description)
{
    size_t    commandLength = strlen("FETCH ALL FROM ") +
                             strlen(cursorName) + 3;
    char      *commandText    = malloc(commandLength);
    PGresult  *result;

```

(continues on next page)

(continued from previous page)

```

int         row;

sprintf(commandText, "FETCH ALL FROM \"%s\"", cursorName);

result = PQexec(conn, commandText);

if (PQresultStatus(result) != PGRES_TUPLES_OK)
fail(conn, PQerrorMessage(conn));

printf("-- %s --\n", description);

for (row = 0; row < PQntuples(result); row++)
{
    const char *delimiter = "\t";
    int         col;

    for (col = 0; col < PQnfields(result); col++)
    {
        printf("%s%s", delimiter, PQgetvalue(result, row, col));
        delimiter = ",";
    }

    printf("\n");
}

PQclear(result);
free(commandText);
}

static void
fail(PGconn *conn, const char *msg)
{
    fprintf(stderr, "%s\n", msg);

    if (conn != NULL)
        PQfinish(conn);

    exit(-1);
}

```

If you call `getEmpsAndDepts(20, 30)`, the server returns a cursor that contains all employees who work in department 20 or 30, and a second cursor containing the description of departments 20 and 30.

```

-- employees --
7369,SMITH,CLERK,7902,17-DEC-80 00:00:00,800.00,,20
7499,ALLEN,SALESMAN,7698,20-FEB-81 00:00:00,1600.00,300.00,30
7521,WARD,SALESMAN,7698,22-FEB-81 00:00:00,1250.00,500.00,30
7566,JONES,MANAGER,7839,02-APR-81 00:00:00,2975.00,,20
7654,MARTIN,SALESMAN,7698,28-SEP-81 00:00:00,1250.00,1400.00,30
7698,BLAKE,MANAGER,7839,01-MAY-81 00:00:00,2850.00,,30
7788,SCOTT,ANALYST,7566,19-APR-87 00:00:00,3000.00,,20
7844,TURNER,SALESMAN,7698,08-SEP-81 00:00:00,1500.00,0.00,30

```

(continues on next page)

(continued from previous page)

```

7876,ADAMS,CLERK,7788,23-MAY-87 00:00:00,1100.00,,20
7900,JAMES,CLERK,7698,03-DEC-81 00:00:00,950.00,,30
7902,FORD,ANALYST,7566,03-DEC-81 00:00:00,3000.00,,20
- - departments - -
20,RESEARCH,DALLAS
30,SALES,CHICAGO

```

5.4 Array binding

EDB Postgres Advanced Server's array binding functionality allows you to send an array of data across the network in a single round-trip. When the back end receives the bulk data, it can use the data to perform insert or update operations.

Perform bulk operations with a prepared statement; use the following function to prepare the statement:

```

PGresult *PQprepare(PGconn *conn,
                    const char *stmtName,
                    const char *query,
                    int nParams,
                    const Oid *paramTypes);

```

Details of PQprepare() can be found in the prepared statement section.

The following functions can be used to perform bulk operations:

- PQBulkStart
- PQexecBulk
- PQBulkFinish
- PQexecBulkPrepared

5.4.1 PQBulkStart

PQBulkStart() initializes bulk operations on the server. You must call this function before sending bulk data to the server. PQBulkStart() initializes the prepared statement specified in stmtName to receive data in a format specified by paramFmts.

5.4.2 API definition

```

PGresult *PQBulkStart(PGconn *conn,
                     const char *Stmt_Name,
                     unsigned int nCol,
                     const int *paramFmts);

```

5.4.3 PQexecBulk

PQexecBulk() is used to supply data (paramValues) for a statement that was previously initialized for bulk operation using PQBulkStart() .

This function can be used more than once after PQBulkStart() to send multiple blocks of data. See the example for more details.

5.4.4 API definition

```
PGresult *PQexecBulk(PGconn *conn,
                     unsigned int nRows,
                     const char *const * paramValues,
                     const int *paramLengths);
```

5.4.5 PQBulkFinish

This function completes the current bulk operation. You can use the prepared statement again without re-preparing it.

5.4.6 API definition

```
PGresult *PQBulkFinish(PGconn *conn);
```

5.4.7 PQexecBulkPrepared

Alternatively, you can use the PQexecBulkPrepared() function to perform a bulk operation with a single function call. PQexecBulkPrepared() sends a request to execute a prepared statement with the given parameters, and waits for the result. This function combines the functionality of PQbulkStart() , PQexecBulk() , and PQBulkFinish() . When using this function, you are not required to initialize or terminate the bulk operation; this function starts the bulk operation, passes the data to the server, and terminates the bulk operation.

Specify a previously prepared statement in the place of stmtName. Commands that are used repeatedly are parsed and planned just once, rather than each time they are executed.

5.4.8 API definition

```
PGresult *PQexecBulkPrepared(PGconn *conn,
                             const char *stmtName,
                             unsigned int nCols,
                             unsigned int nRows,
                             const char *const *paramValues,
                             const int *paramLengths,
                             const int *paramFormats);
```

5.4.9 Example code (Using PQBulkStart, PQexecBulk, PQBulkFinish)

The following example uses PQBulkStart , PQexecBulk , and PQBulkFinish .

```
void InsertDataUsingBulkStyle( PGconn *conn )
{
    PGresult      *res;
    Oid            paramTypes[2];
    char          *paramVals[5][2];
    int            paramLens[5][2];
    int            paramFmts[2];
    int            i;

    int            a[5] = { 10, 20, 30, 40, 50 };
    char           b[5][10] = { "Test_1", "Test_2", "Test_3",
    "Test_4", "Test_5" };

    paramTypes[0] = 23;
    paramTypes[1] = 1043;
    res = PQprepare( conn, "stmt_1", "INSERT INTO testtable1 values( $1, $2
)", 2, paramTypes );
    PQclear( res );

    paramFmts[0] = 1;    /* Binary format */
    paramFmts[1] = 0;

    for( i = 0; i < 5; i++ )
    {
        a[i] = htonl( a[i] );
        paramVals[i][0] = &(a[i]);
        paramVals[i][1] = b[i];

        paramLens[i][0] = 4;
        paramLens[i][1] = strlen( b[i] );
    }

    res = PQBulkStart(conn, "stmt_1", 2, paramFmts);
    PQclear( res );
    printf( "< -- PQBulkStart -- >\n" );

    res = PQexecBulk(conn, 5, (const char *const *)paramVals, (const
int*)paramLens);
    PQclear( res );
    printf( "< -- PQexecBulk -- >\n" );

    res = PQexecBulk(conn, 5, (const char *const *)paramVals, (const
int*)paramLens);
    PQclear( res );
    printf( "< -- PQexecBulk -- >\n" );

    res = PQBulkFinish(conn);
    PQclear( res );
    printf( "< -- PQBulkFinish -- >\n" );
}
```

5.4.10 Example code (Using PQexecBulkPrepared)

The following example uses PQexecBulkPrepared .

```
void InsertDataUsingBulkStyleCombinedVersion( PGconn *conn )
{
    PGresult      *res;
    Oid            paramTypes[2];
    char          *paramVals[5][2];
    int            paramLens[5][2];
    int            paramFmts[2];
    int            i;

    int            a[5] = { 10, 20, 30, 40, 50 };
    char          b[5][10] = { "Test_1", "Test_2", "Test_3",
    "Test_4", "Test_5" };

    paramTypes[0] = 23;
    paramTypes[1] = 1043;
    res = PQprepare( conn, "stmt_2", "INSERT INTO testtable1 values(
    $1, $2)", 2, paramTypes );
    PQclear( res );

    paramFmts[0] = 1;    /* Binary format */
    paramFmts[1] = 0;

    for( i = 0; i < 5; i++ )
    {
        a[i] = htonl( a[i] );
        paramVals[i][0] = &(a[i]);
        paramVals[i][1] = b[i];

        paramLens[i][0] = 4;
        paramLens[i][1] = strlen( b[i] );
    }

    res = PQexecBulkPrepared(conn, "stmt_2", 2, 5, (const char *const
    *)paramVals, (const int *)paramLens, (const int *)paramFmts);
    PQclear( res );
}
```

DEBUGGER

The Debugger gives developers and DBAs the ability to test and debug server-side programs using a graphical, dynamic environment. The types of programs that can be debugged are SPL stored procedures, functions, triggers, and packages as well as PL/pgSQL functions and triggers.

The Debugger is integrated with *pgAdmin 4* and *EDB Postgres Enterprise Manager*. If you have installed EDB Postgres Advanced Server on a Windows host, pgAdmin 4 is automatically installed; the pgAdmin 4 icon is in the Windows Start menu. If your EDB Postgres Advanced Server host is on a CentOS or Linux system, you can use `yum` to install pgAdmin4. Open a command line, assume superuser privileges, and enter:

```
yum install edb-pgadmin4*
```

On Linux, the `edb-asxx-server-pldebugger` RPM package where `xx` is the EDB Postgres Advanced Server version number, must be installed as well. Information about pgAdmin 4 is available at:

<https://www.pgadmin.org/>

The RPM installation adds the pgAdmin4 icon to your Applications menu.

There are two basic ways the Debugger can be used to test programs:

- ****StandaloneDebugging.**** The Debugger **is** used to start the program to be tested.
 - You supply **any input** parameter values required by the program **and** you can
 - immediately observe **and** step through the code of the program. Standalone
 - debugging **is** the typical method used **for** new programs **and for** initial problem
 - investigation.
- ****In-ContextDebugging.**** The program to be tested **is** initiated by an application
 - other than the Debugger. You first **set** a ***global breakpoint*** on the program to be
 - tested. The application that makes the first call to the program encounters the
 - **global breakpoint**. The application suspends execution at which point the Debugger
 - takes control of the called program. You can then observe **and** step through the
 - code of the called program **as** it runs within the context of the calling
 - application. After you have completely stepped through the code of the called
 - program **in** the Debugger, the suspended application resumes execution. In-context
 - debugging **is** useful **if** it **is** difficult to reproduce a problem using standalone
 - debugging due to **complex** interaction **with** the calling application.

The debugging tools and operations are the same whether using standalone or in-context debugging. The difference is in how the program to be debugged is invoked.

The following sections discuss the features and functionality of the Debugger using the standalone debugging method. The directions for starting the Debugger for in-context debugging are discussed in the

Setting Global Breakpoint for In-Context Debugging .

6.1 Configuring the Debugger

Before using the Debugger, edit the `postgresql.conf` file (located in the data subdirectory of your EDB Postgres Advanced Server home directory), adding `$libdir/plugin_debugger` to the libraries listed in the `shared_preload_libraries` configuration parameter:

```
shared_preload_libraries = $libdir/dbms_pipe,$libdir/edb_gen,$libdir/plugin_debugger
```

- On Linux, the `postgresql.conf` file is located in: `/var/lib/edb/asxx/data`
- On Windows, the `postgresql.conf` file is located in: `C:\Program Files\edb\asxx\data`

Where **xx** is the version of EDB Postgres Advanced Server.

After modifying the `shared_preload_libraries` parameter, you must restart the database server.

6.2 Starting the Debugger

Use pgAdmin 4 to access the Debugger for standalone debugging. To open the Debugger, highlight the name of the stored procedure or function you wish to debug in the pgAdmin 4 Browser panel. Then, navigate through the Object menu to the Debugging menu and select Debug from the submenu.

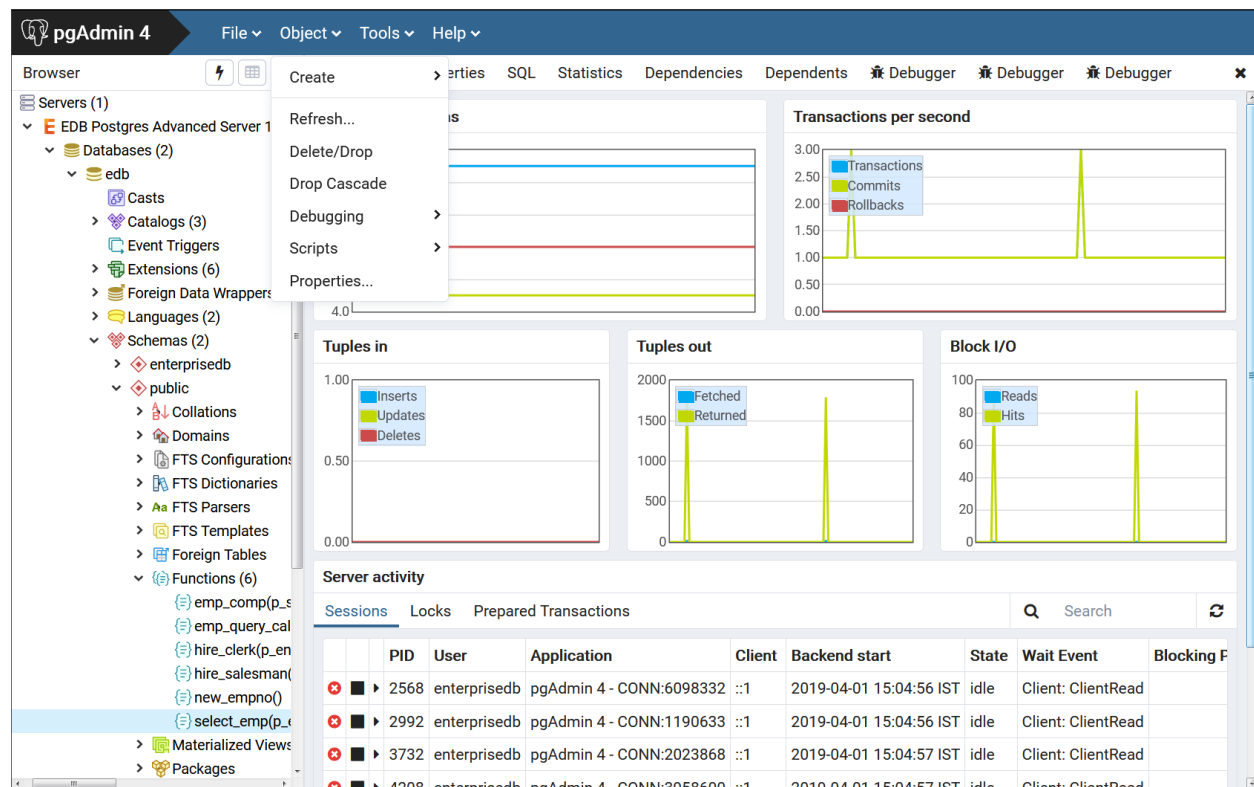


Fig. 1: Starting the Debugger from the Object menu

Fig. 1: Starting the Debugger from the Object menu

You can also right-click on the name of the stored procedure or function in the pgAdmin 4 Browser , and select **Debugging** , and the **Debug** from the context menu.

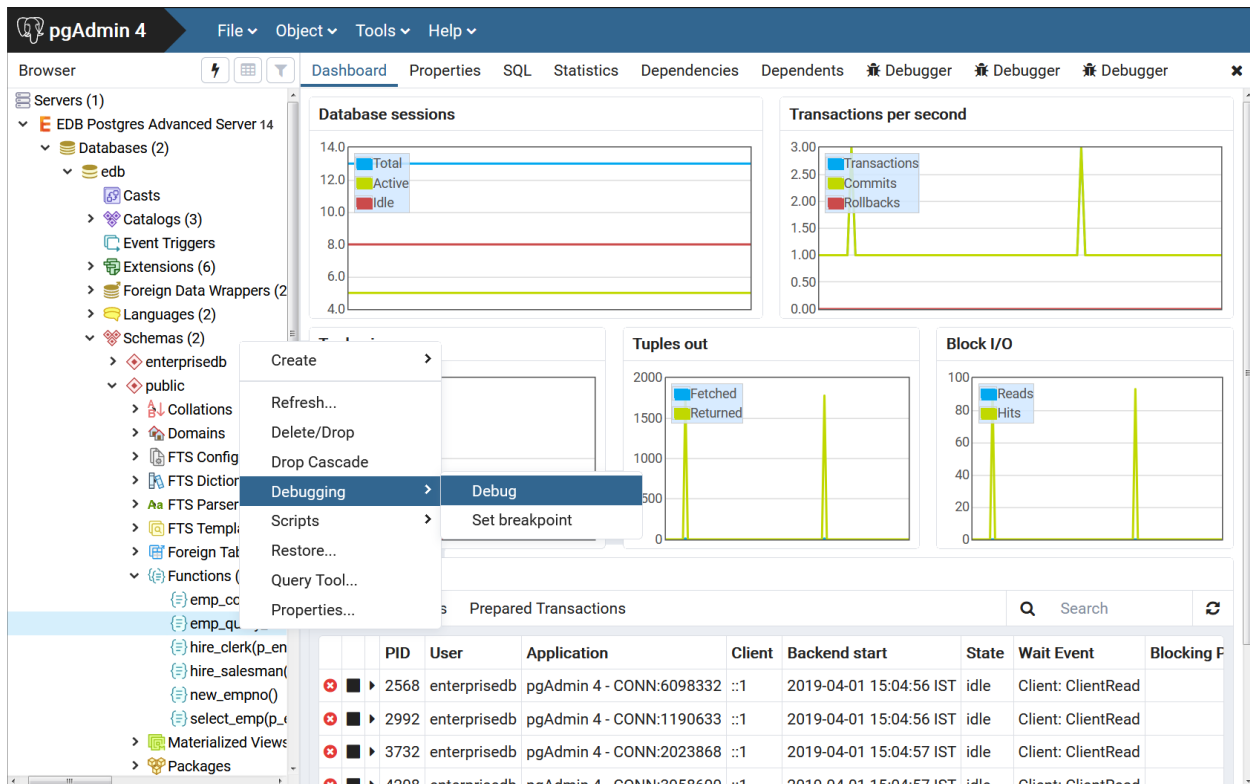


Fig. 2: Starting the Debugger from the object's context menu

Fig. 2: Starting the Debugger from the Object's context menu

Note that triggers cannot be debugged using standalone debugging. Triggers must be debugged using in-context debugging. See the

Setting Global Breakpoint for In-Context Debugging for information on setting a global breakpoint for in-context debugging.

To debug a package, highlight the specific procedure or function under the package node of the package you wish to debug and follow the same directions as for stored procedures and functions.

6.3 The Debugger window

You can use the Debugger window to pass parameter values when you are standalone-debugging a program that expects parameters. When you start the debugger, the Debugger window opens automatically to display any IN or IN OUT parameters expected by the program. If the program declares no IN or IN OUT parameters, the Debugger window does not open.

Fig. 3: The Debugger window

Use the fields on the Debugger window to provide a value for each parameter:

- The Name field contains the formal parameter name.
- The Type field contains the parameter data type.
- Check the Null? checkbox to indicate that the parameter is a NULL value.

Debugger						
Name	Type	Null?	Expression?	Value	Use Default?	Default value
p_sal	numeric	☒	☒	NaN	☐	<No default value>
p_comm	numeric	☒	☒	NaN	☐	<No default value>

✕ Cancel
🐞 Debug

Fig. 3: The Debugger window

- Check the Expression? checkbox if the Value field contains an expression.
- The Value field contains the parameter value that is passed to the program.
- Check the Use Default? checkbox to indicate that the program should use the value in the Default Value field.
- The Default Value field contains the default value of the parameter.

Press the Tab key to select the next parameter in the list for data entry, or click on a Value field to select the parameter for data entry.

If you are debugging a procedure or function that is a member of a package that has an initialization section, check the Debug Package Initializer check box to instruct the Debugger to step into the package initialization section, allowing you to debug the initialization section code before debugging the procedure or function. If you don't select the check box, the Debugger executes the package initialization section without allowing you to see or step through the individual lines of code as they are executed.

After entering the desired parameter values, click the Debug button to start the debugging process. Click the Cancel button to terminate the Debugger.

Note: The *Debugger* window does not open during in-context debugging. Instead, the application calling the program to be debugged must supply any required input parameter values.

When you have completed a full debugging cycle by stepping through the program code, the Debugger window reopens, allowing you to enter new parameter values and repeat the debugging cycle, or end the debugging session.

6.4 Main Debugger window

The Main Debugger window contains two panels:

- The top Program Body panel displays the program source code.
- The bottom Tabs panel provides a set of tabs for different information.

Use the Tool Bar icons located at the top panel to access debugging functions.

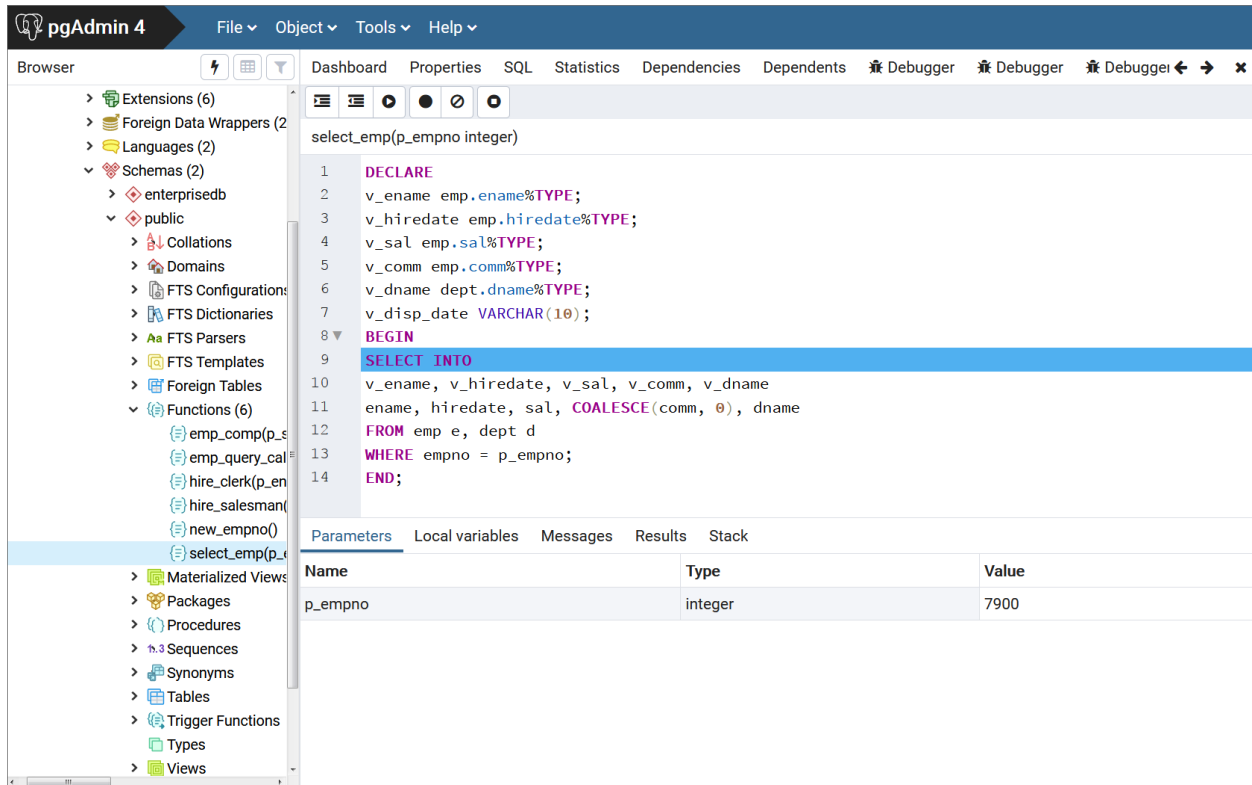


Fig. 4: The Main Debugger window

Fig. 4: The Main Debugger window

The two panels are described in the following sections.

6.4.1 The Program Body panel

The Program Body panel displays the source code of the program that is being debugged.

Fig. 5: The Program Body

The figure shows that the Debugger is about to execute the SELECT statement. The blue indicator in the program body highlights the next statement to execute.

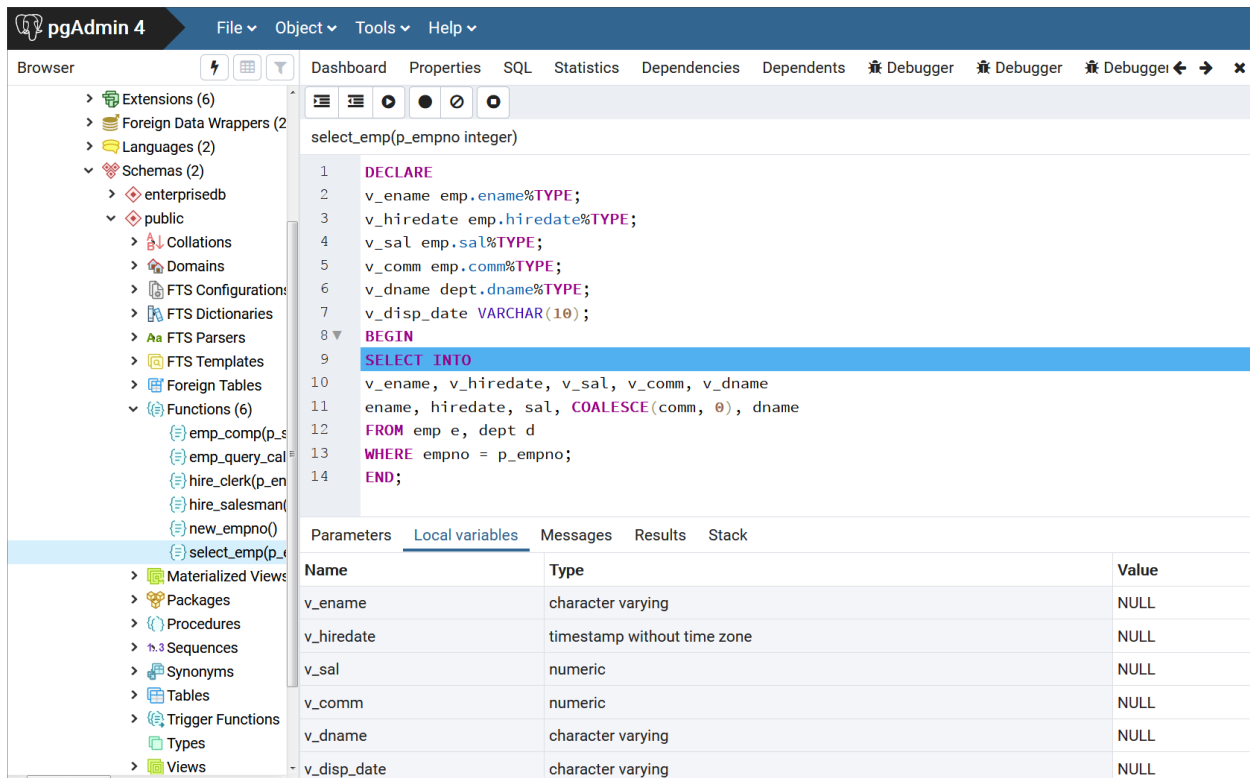


Fig. 5: The Program Body

6.4.2 The Tabs panel

You can use the bottom Tabs panel to view or modify parameter values or local variables, or to view messages generated by `RAISE INFO` and function results.

The following is the information displayed by the tabs in the panel:

- The **Parameters** tab displays the current parameter values.
- The **Local variables** tab displays the value of any variables declared within the program.
- The **Messages** tab displays any results returned by the program as it executes.
- The **Results** tab displays program results (if applicable) such as the value from the `RETURN` statement of a function.
- The **Stack** tab displays the call stack.

The following figures show the results from the various tabs.

Fig. 6: The Parameters tab

Fig. 7: The Local variables tab

Fig. 8: The Messages tab

Fig. 9: The Results tab

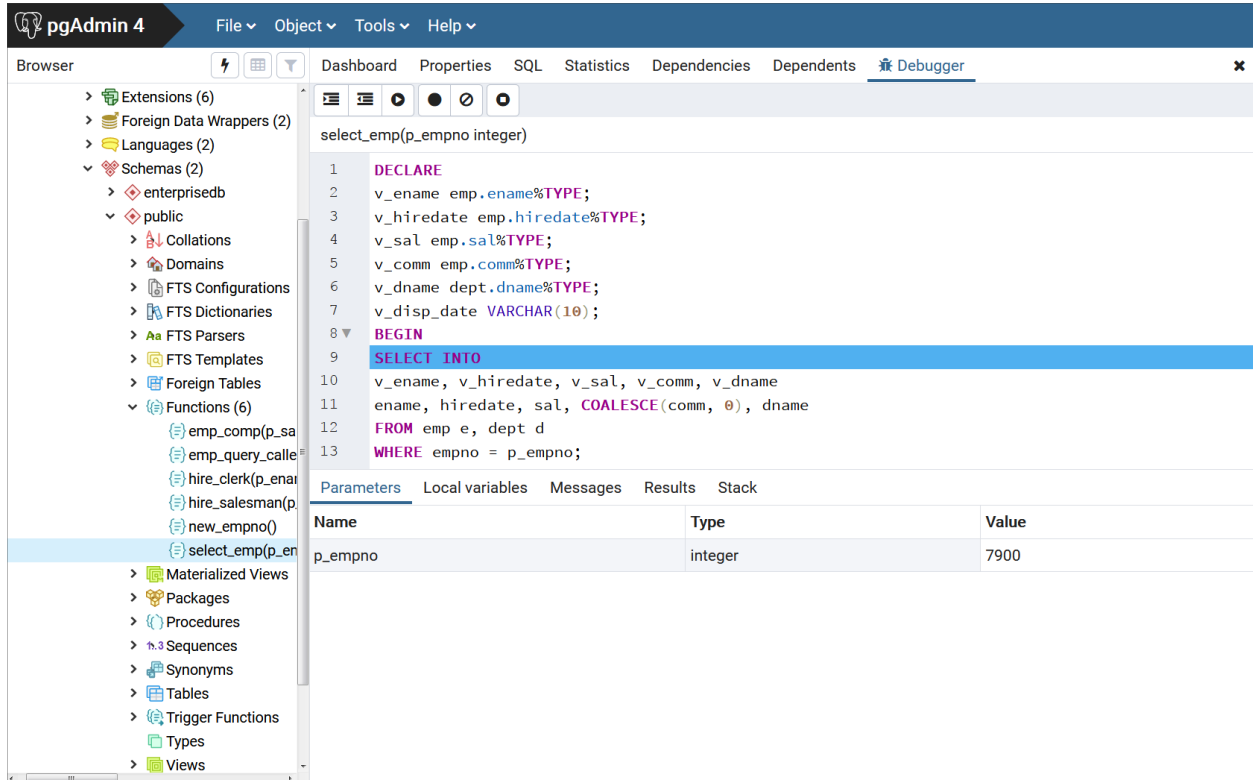


Fig. 6: The Parameters tab

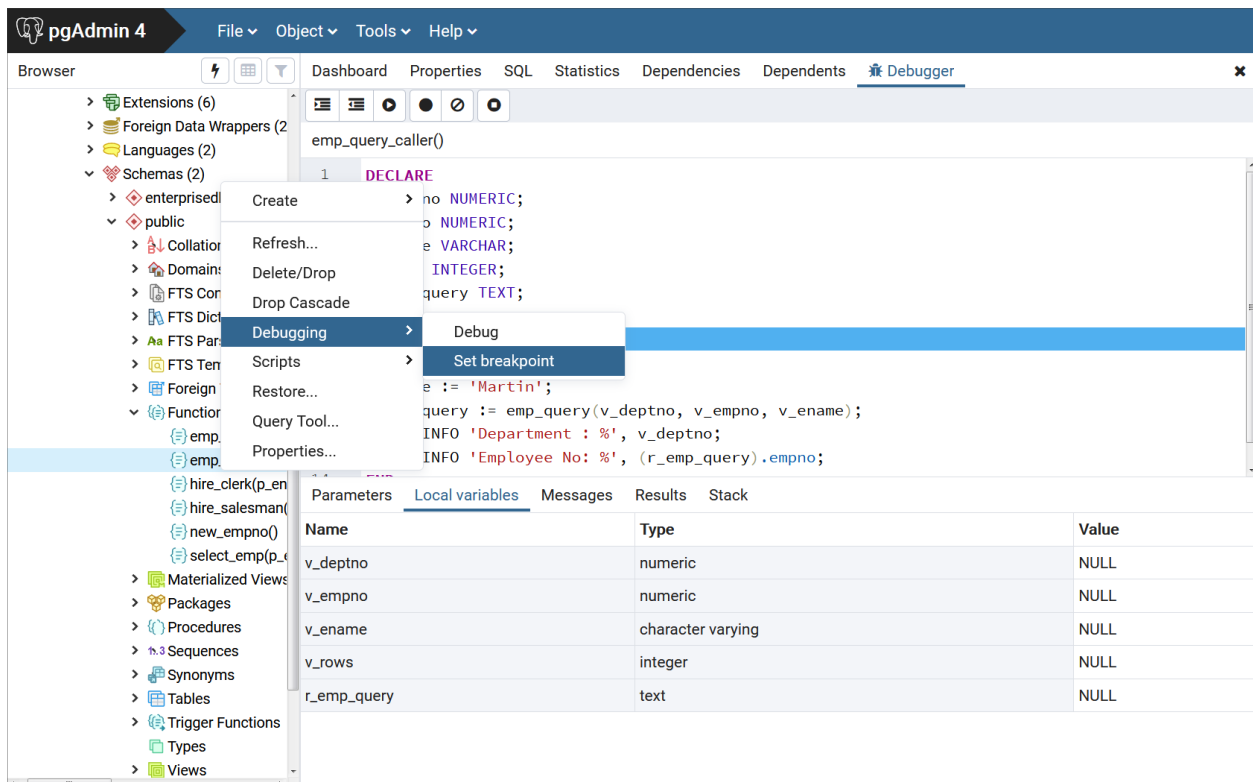


Fig. 7: The Local variables tab

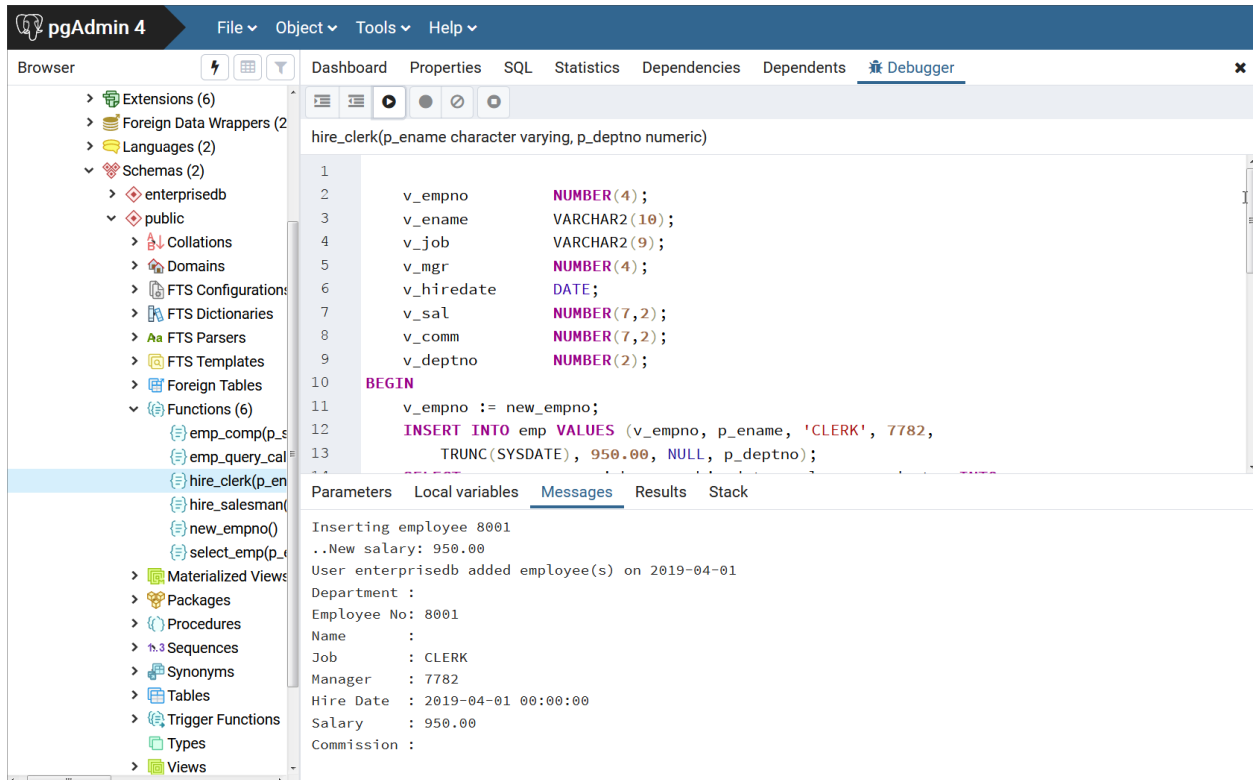


Fig. 8: The Messages tab

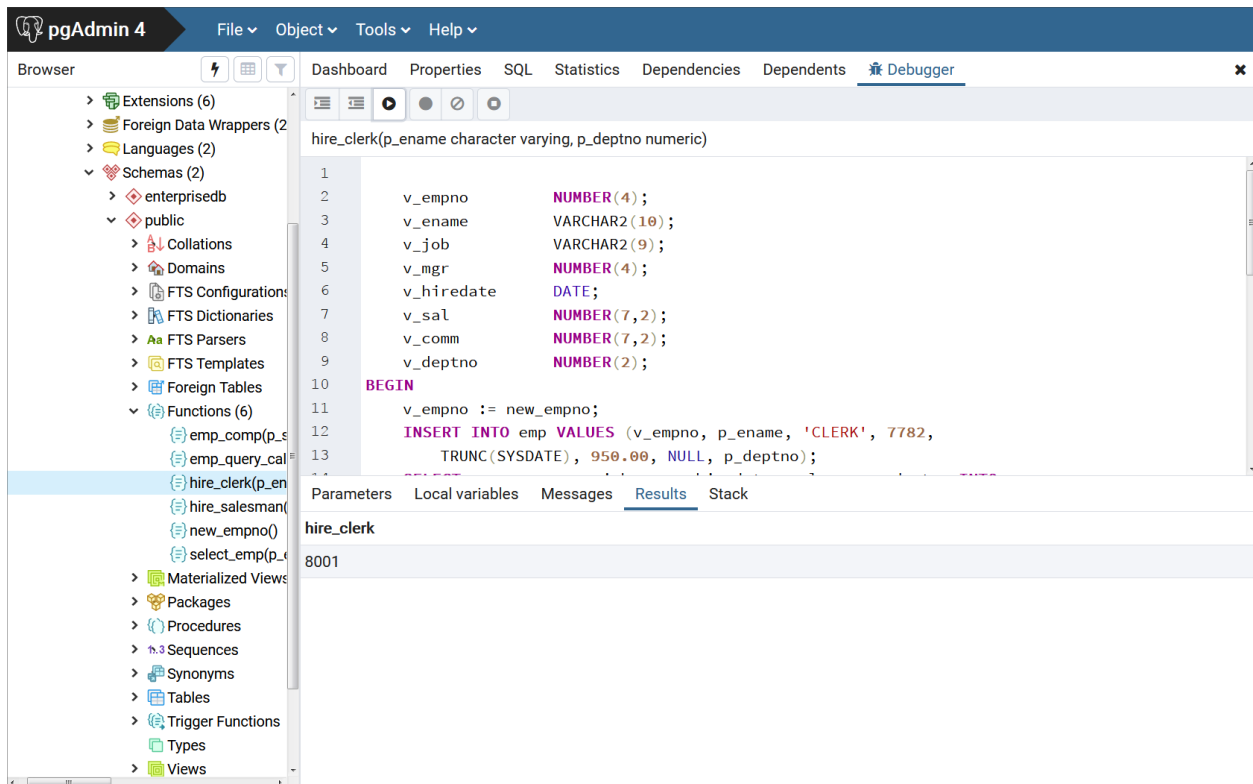


Fig. 9: The Results tab

6.4.3 The Stack tab

The **Stack** tab displays a list of programs that are currently on the call stack (programs that have been invoked, but which have not yet completed). When a program is called, the name of the program is added to the top of the list displayed in the Stack tab. When the program ends, its name is removed from the list.

The Stack tab also displays information about program calls. The information includes:

- The location of the call within the program
- The call arguments
- The name of the program being called

Reviewing the call stack can help you trace the course of execution through a series of nested programs.

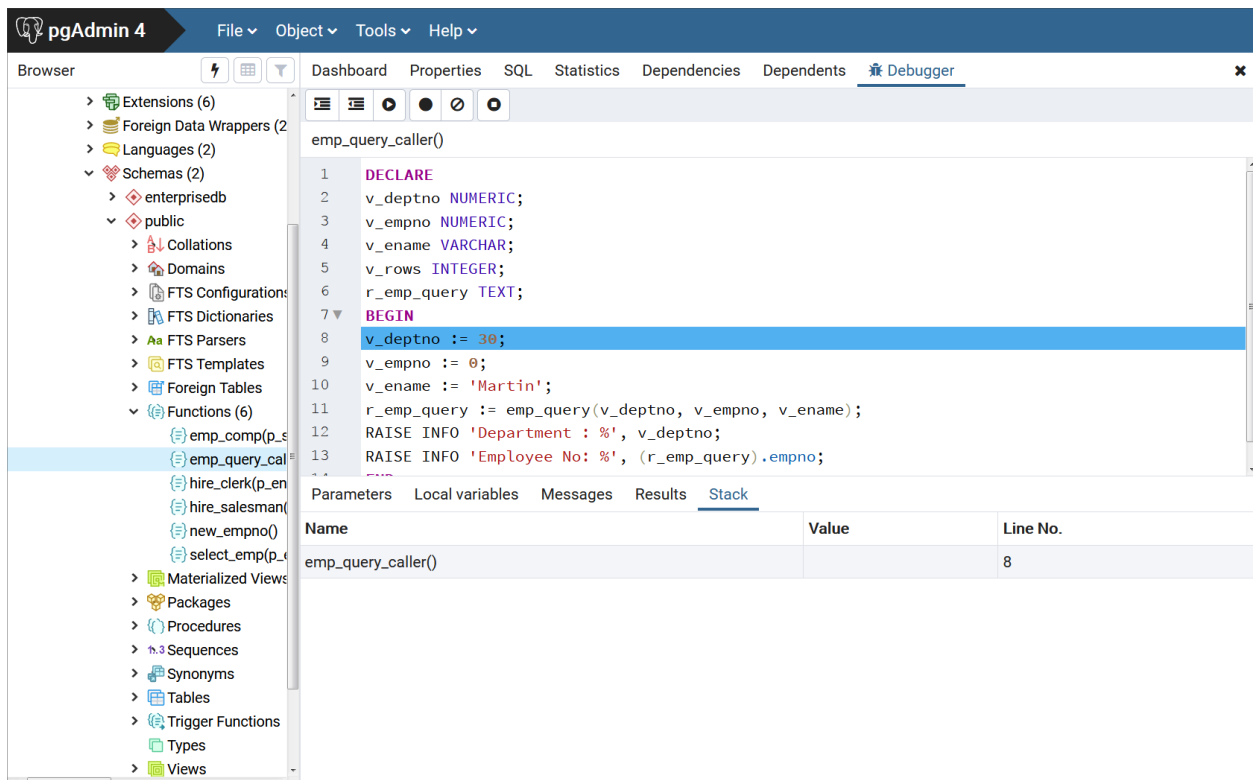


Fig. 10: A debugged program calling a subprogram

Fig. 10: A debugged program calling a subprogram

The above figure shows that `emp_query_caller` is about to call a subprogram named `emp_query`. `emp_query_caller` is currently at the top of the call stack.

After the call to `emp_query` executes, `emp_query` is displayed at the top of the Stack tab, and its code is displayed in the Program Body panel.

Fig. 11: Debugging the called subprogram

Upon completion of execution of the subprogram, control returns to the calling program (`emp_query_caller`), now displayed at the top of the Stack tab.

Fig. 12: Control returns from debugged subprogram

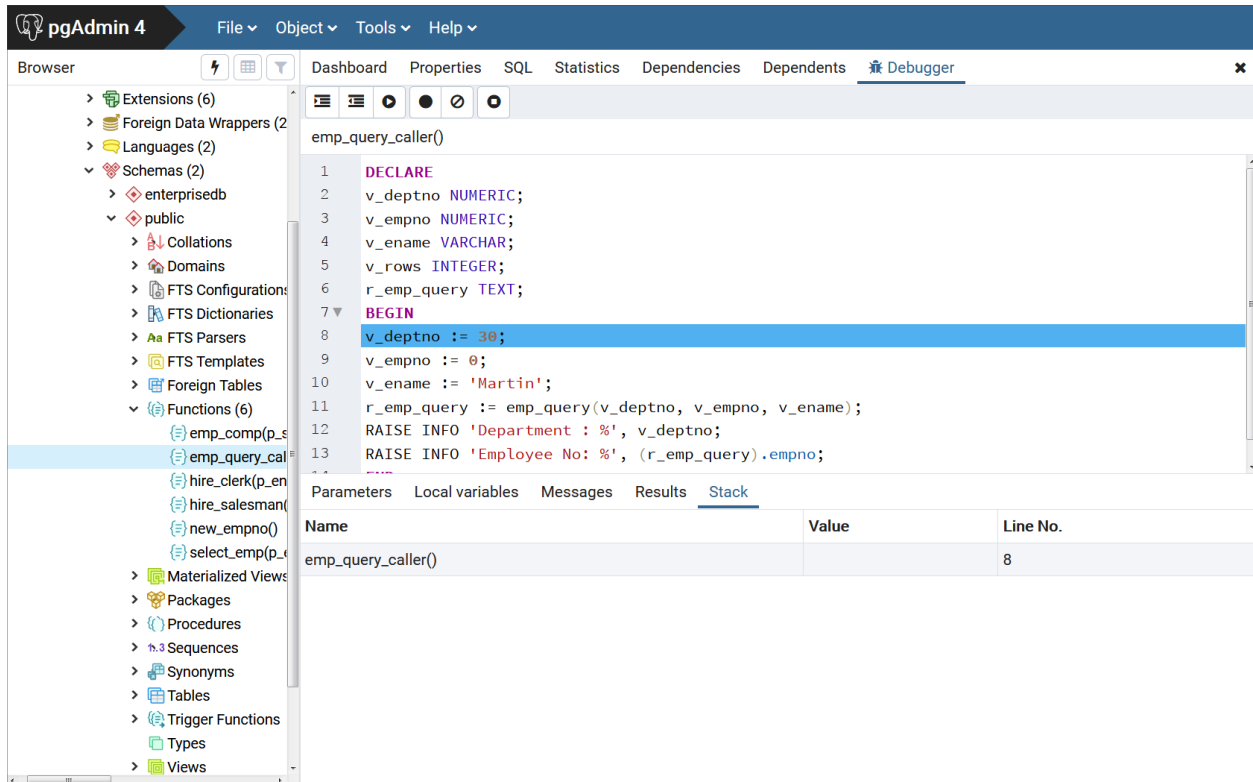


Fig. 11: Debugging the called subprogram

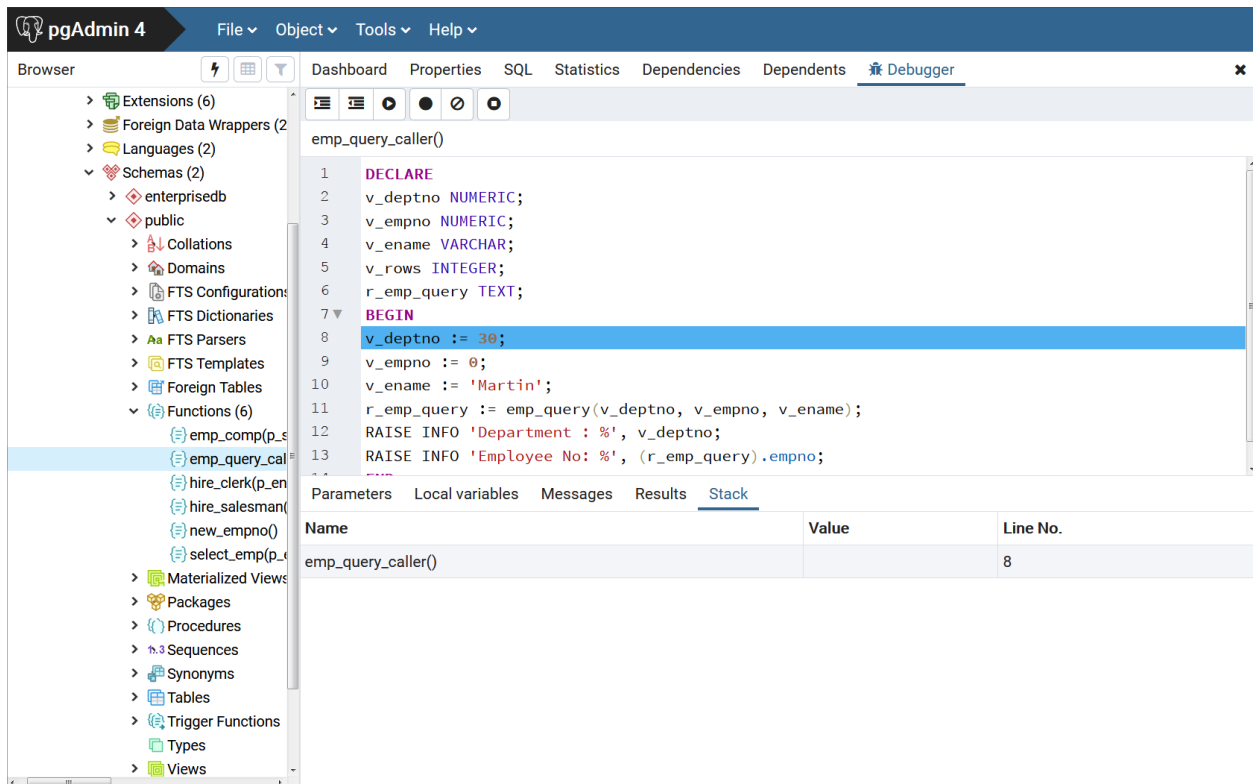


Fig. 12: Control returns from debugged subprogram

6.5 Debugging a program

You can perform the following operations to debug a program:

- Step through the program one line at a time
- Execute the program until you reach a breakpoint
- View and change local variable values within the program

6.5.1 Stepping through the code

Use the tool bar icons to step through a program with the Debugger:

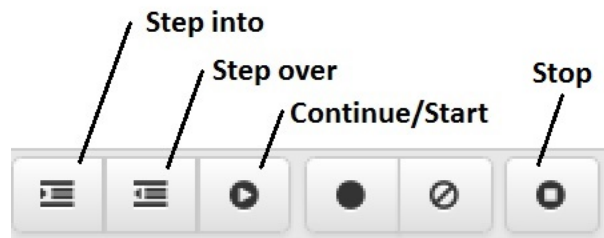


Fig. 13: The Tool bar icons

Fig. 13: The Tool bar icons

The icons serve the following purposes:

- ****Stepinto.**** Click the `Step into` icon to execute the currently highlighted line of code.
- ****Stepover.**** Click the `Step over` icon to execute a line of code, stepping over any sub-functions invoked by the code. The sub-function executes, but is not debugged unless it contains a breakpoint.
- ****Continue/Start.**** Click the `Continue/Start` icon to execute the highlighted code, and continue until the program encounters a breakpoint or completes.
- ****Stop.**** Click the `Stop` icon to halt the execution of a program.

6.5.2 Using breakpoints

As the Debugger executes a program, it pauses whenever it reaches a breakpoint. When the Debugger pauses, you can observe or change local variables, or navigate to an entry in the call stack to observe variables or set other breakpoints. The next step into, step over, or continue operation forces the debugger to resume execution with the next line of code following the breakpoint. There are two types of breakpoints:

Local breakpoint - A local breakpoint can be set at any executable line of code within a program. The Debugger pauses execution when it reaches a line where a local breakpoint has been set.

Global breakpoint - A global breakpoint triggers when *any* session reaches that breakpoint. Set a global breakpoint if you want to perform in-context debugging of a program. When a global breakpoint is set on a program, the debugging

session that set the global breakpoint waits until that program is invoked in another session. A global breakpoint can only be set by a superuser.

To create a local breakpoint, left-click within the grey shaded margin to the left of the line of code where you want the local breakpoint set. Where you click in the grey shaded margin should be close to the right side of the margin as in the spot where the breakpoint dot is shown on source code line 12.

When created, the Debugger displays a dark dot in the margin, indicating a breakpoint has been set at the selected line of code.

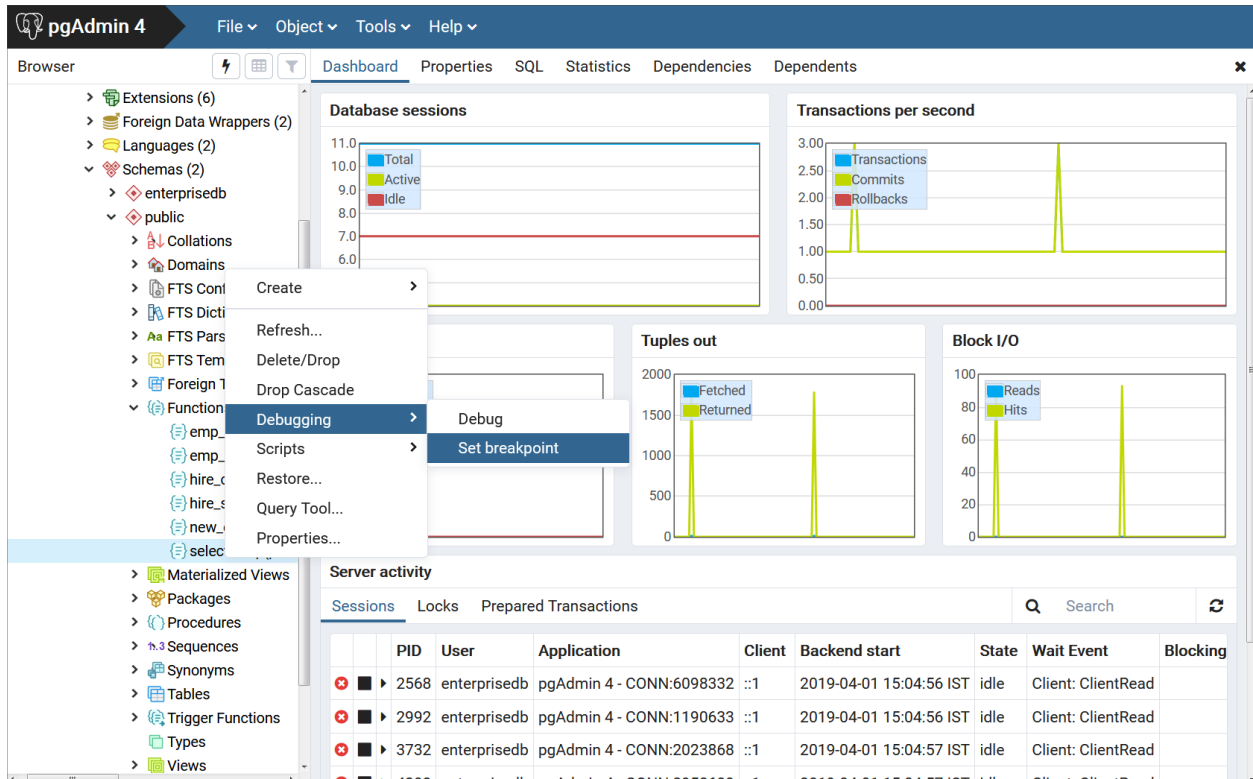


Fig. 14: Set a breakpoint by clicking in left-hand margin

Fig. 14: Set a breakpoint by clicking in left-hand margin

You can set as many local breakpoints as desired. Local breakpoints remain in effect for the duration of a debugging session until they are removed.

6.5.3 Removing a local breakpoint

To remove a local breakpoint, left-click the mouse on the breakpoint dot in the grey shaded margin of the Program Body panel. The dot disappears, indicating that the breakpoint has been removed.

You can remove all of the breakpoints from the program that currently appears in the Program Body frame by clicking the Clear all breakpoints icon.

Fig. 15: Clear all breakpoints icon

Note: When you perform any of the preceding actions, only the breakpoints in the program that currently appears in the *Program Body* panel are removed. Breakpoints in called subprograms or breakpoints in programs that call the

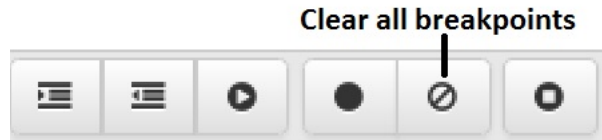


Fig. 15: Clear all breakpoints icon

program currently appearing in the *Program Body* panel are not removed.

6.5.4 Setting a global breakpoint for in-context debugging

To set a global breakpoint for in-context debugging, highlight the stored procedure, function, or trigger on which you wish to set the breakpoint in the *Browser* panel. Navigate through the *Object* menu to select *Debugging*, and then *Set Breakpoint*.

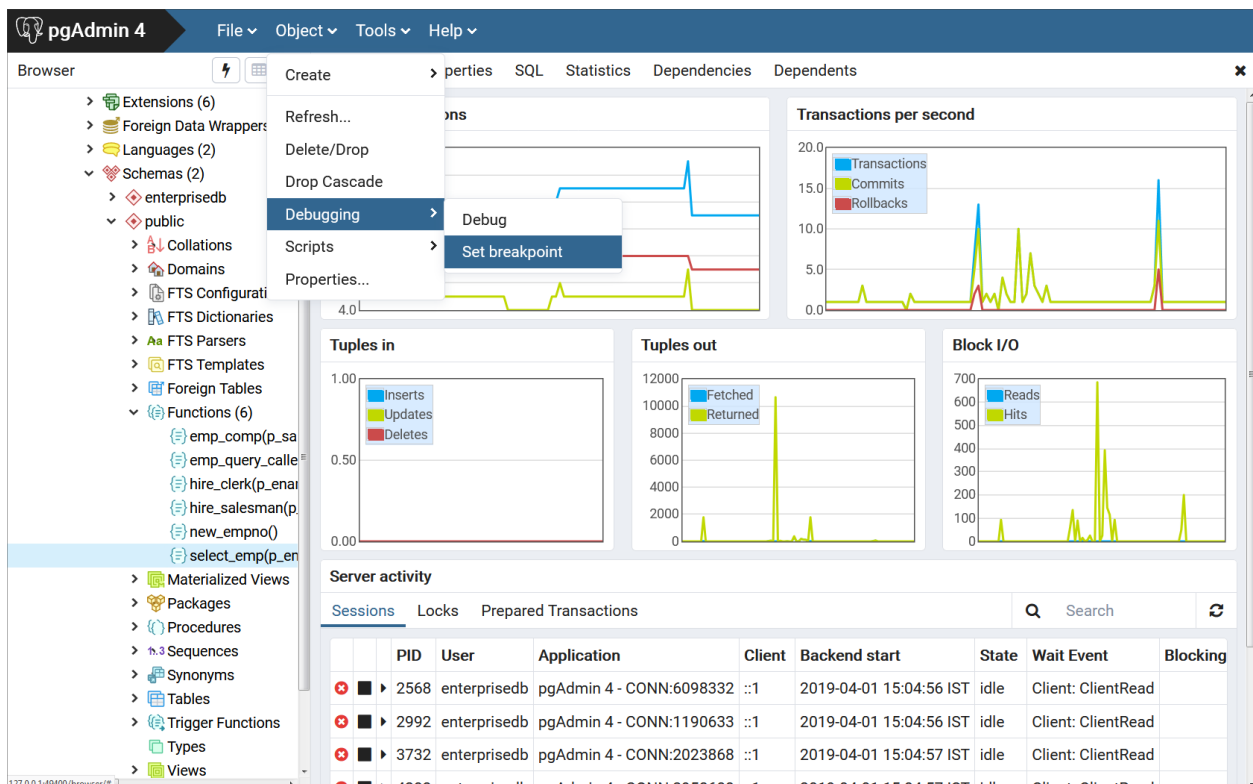


Fig. 16: Setting a global breakpoint from the Object menu

Fig. 16: Setting a global breakpoint from the Object menu

Alternatively, you can right-click on the name of the stored procedure, function, or trigger on which you wish to set a global breakpoint and select *Debugging*, then *Set Breakpoint* from the context menu as shown by the following.

Fig. 17: Setting a global breakpoint from the Object's context menu

To set a global breakpoint on a trigger, expand the table node that contains the trigger, highlight the specific trigger you wish to debug, and follow the same directions as for stored procedures and functions.

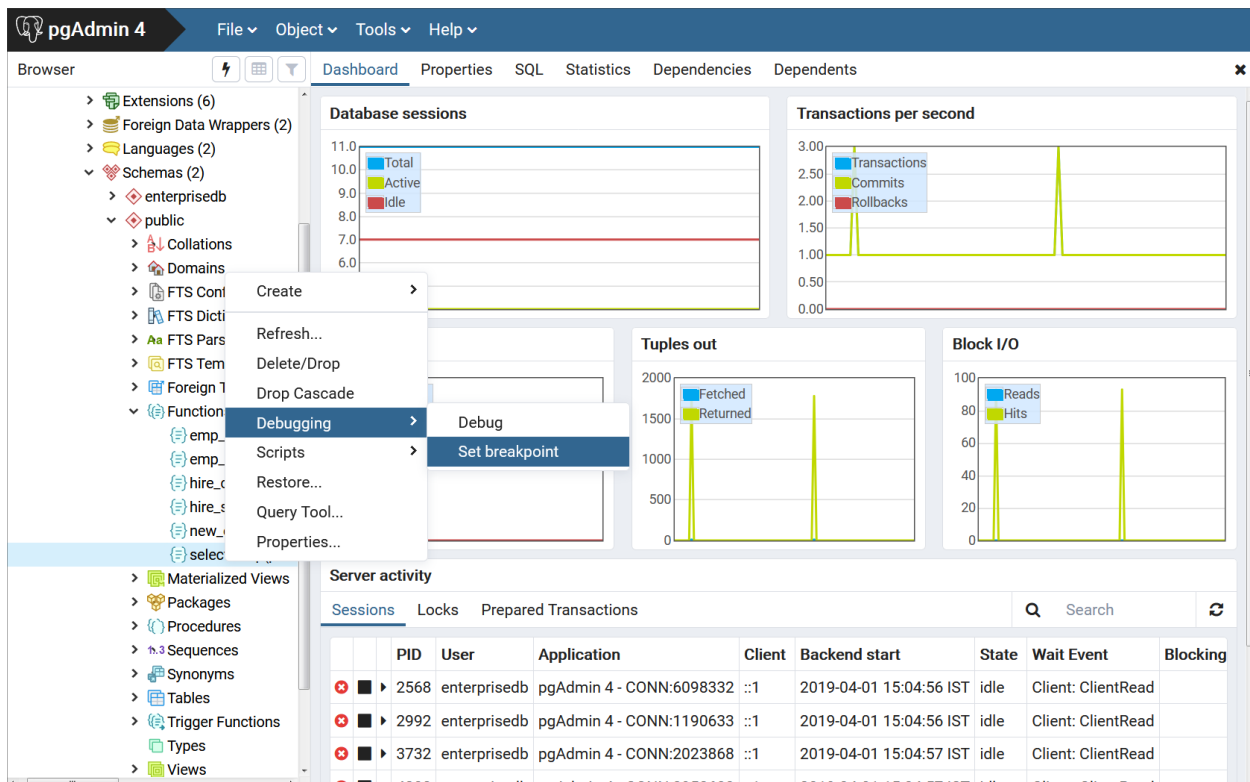


Fig. 17: Setting a global breakpoint from the object's context menu

To set a global breakpoint in a package, highlight the specific procedure or function under the package node of the package you wish to debug and follow the same directions as for stored procedures and functions.

After you choose **Set Breakpoint**, the Debugger window opens and waits for an application to call the program to be debugged.

Fig. 18: Waiting for invocation of program to be debugged

The PSQL client invokes the `select_emp` function (on which a global breakpoint has been set).

```
$ psql edb enterprisedb
psql.bin (14.0.0, server 14.0.0)
Type "help" for help.

edb=# SELECT select_emp(7900);
```

The `select_emp` function does not complete until you step through the program in the Debugger.

Fig. 19: Program on which a global breakpoint has been set

You can now debug the program using any of the previously discussed operations such as step into, step over, and continue, or set local breakpoints. When you have stepped through execution of the program, the calling application (PSQL) regains control and the `select_emp` function completes execution and its output is displayed.

```
$ psql edb enterprisedb
psql.bin (14.0.0, server 14.0.0)
Type "help" for help.
```

(continues on next page)

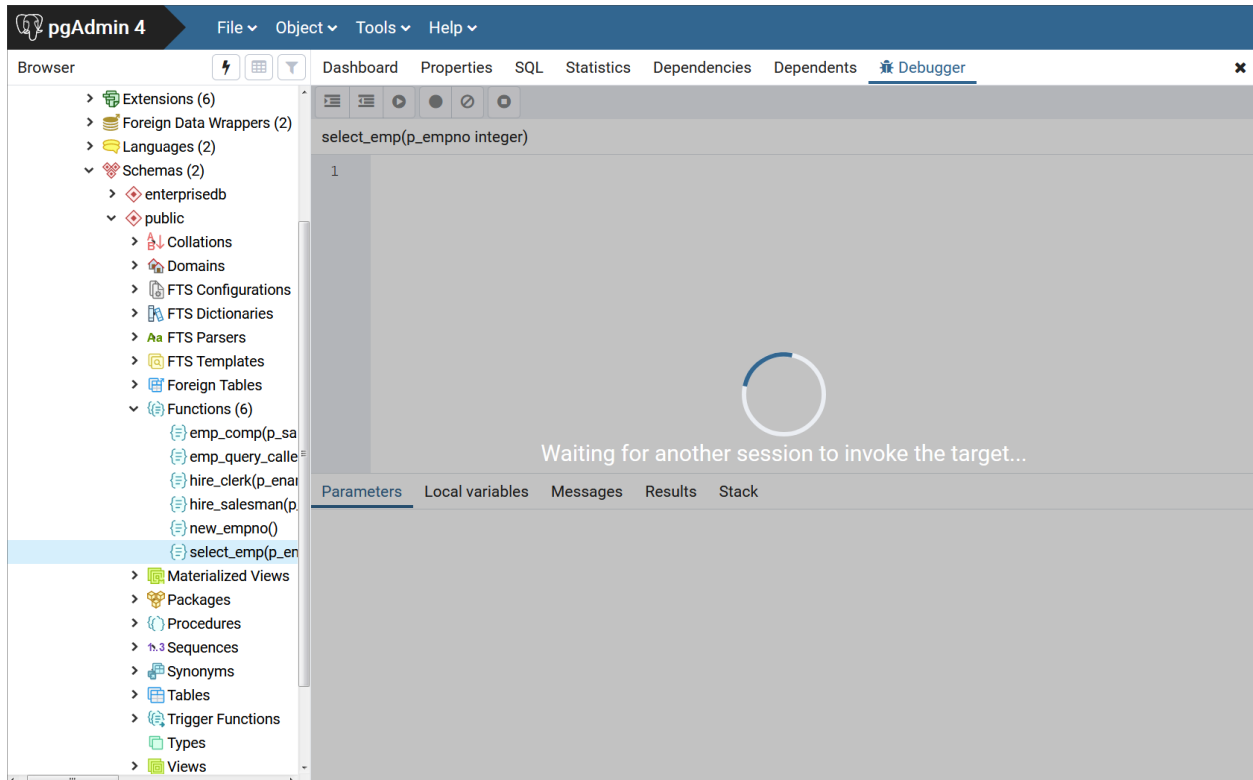


Fig. 18: Waiting for invocation of program to be debugged

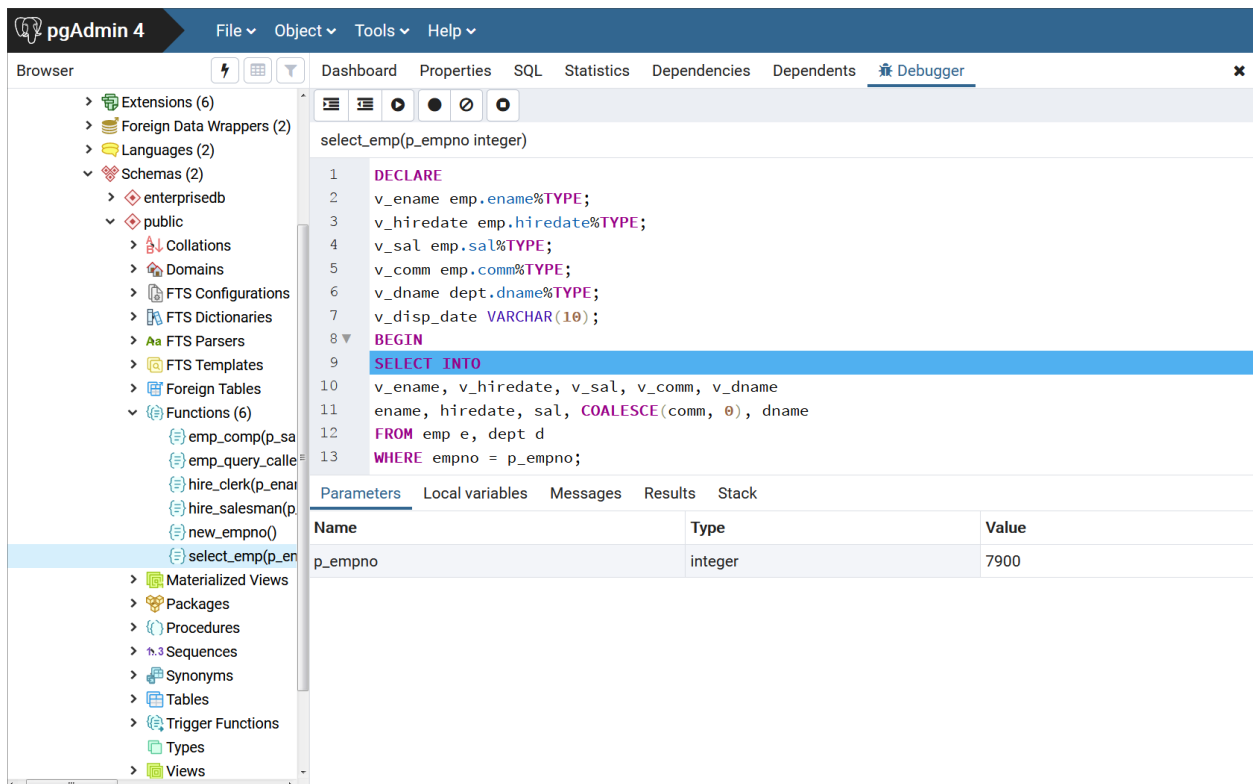


Fig. 19: Program on which a global breakpoint has been set

(continued from previous page)

```

edb=# SELECT select_emp(7900);
__OUTPUT__
INFO: Number      : 7900
INFO: Name        : JAMES
INFO: Hire Date   : 12/03/1981
INFO: Salary      : 950.00
INFO: Commission: 0.00
INFO: Department: SALES
select_emp
- - - - -
(1 row)

```

At this point, you can end the Debugger session. If you don't end the Debugger session, the next application that invokes the program encounters the global breakpoint and the debugging cycle begins again.

6.5.5 Exiting the Debugger

To end a Debugger session and exit the Debugger, click on the close icon (x) located in the upper-right corner to close the tab.



Fig. 20: Exiting from the Debugger

Fig. 20: Exiting from the Debugger

PERFORMANCE ANALYSIS AND TUNING

EDB Postgres Advanced Server provides various tools for performance analysis and tuning. These features are described in this section.

7.1 Dynatune

EDB Postgres Advanced Server supports dynamic tuning of the database server to make the optimal usage of the system resources available on the host machine on which it is installed. The two parameters that control this functionality are located in the `postgresql.conf` file. These parameters are:

- `edb_dynatune`
- `edb_dynatune_profile`

7.1.1 `edb_dynatune`

`edb_dynatune` determines how much of the host system's resources are to be used by the database server based upon the host machine's total available resources and the intended usage of the host machine.

When EDB Postgres Advanced Server is initially installed, the `edb_dynatune` parameter is set in accordance with the selected usage of the host machine on which it was installed - i.e., development machine, mixed use machine, or dedicated server. For most purposes, there is no need for the database administrator to adjust the various configuration parameters in the `postgresql.conf` file in order to improve performance.

You can change the value of the `edb_dynatune` parameter after the initial installation of EDB Postgres Advanced Server by editing the `postgresql.conf` file. The postmaster must be restarted in order for the new configuration to take effect.

The `edb_dynatune` parameter can be set to any integer value between 0 and 100, inclusive. A value of 0, turns off the dynamic tuning feature thereby leaving the database server resource usage totally under the control of the other configuration parameters in the `postgresql.conf` file.

A low non-zero, value (e.g., 1 - 33) dedicates the least amount of the host machine's resources to the database server. This setting would be used for a development machine where many other applications are being used.

A value in the range of 34 - 66 dedicates a moderate amount of resources to the database server. This setting might be used for a dedicated application server that may have a fixed number of other applications running on the same machine as EDB Postgres Advanced Server.

The highest values (e.g., 67 - 100) dedicate most of the server's resources to the database server. This setting would be used for a host machine that is totally dedicated to running EDB Postgres Advanced Server.

Once a value of `edb_dynatune` is selected, database server performance can be further fine-tuned by adjusting the other configuration parameters in the `postgresql.conf` file. Any adjusted setting overrides the corresponding value

chosen by `edb_dynatune`. You can change the value of a parameter by un-commenting the configuration parameter, specifying the desired value, and restarting the database server.

7.1.2 edb_dynatune_profile

The `edb_dynatune_profile` parameter is used to control tuning aspects based upon the expected workload profile on the database server. This parameter takes effect upon startup of the database server.

The possible values for `edb_dynatune_profile` are:

Value	Usage
<i>oltp</i>	Recommended when the database server is processing heavy online transaction processing workloads.
<i>reporting</i>	Recommended for database servers used for heavy data reporting.
<i>mixed</i>	Recommended for servers that provide a mix of transaction processing and data reporting.

7.2 EDB wait states

The *EDB wait states* contrib module contains two main components.

7.2.1 EDB Wait States Background Worker (EWSBW)

When the wait states background worker is registered as one of the shared preload libraries, EWSBW probes each of the running sessions at regular intervals.

For every session it collects information such as the database to which it is connected, the logged in user of the session, the query running in that session, and the wait events on which it is waiting.

This information is saved in a set of files in a user-configurable path and directory folder given by the `edb_wait_states.directory` parameter to be added to the `postgresql.conf` file. The specified path must be a full, absolute path and not a relative path.

The following describes the installation process on a Linux system.

Step1: EDB wait states is installed with the `edb-asxx-server-edb_wait_states` RPM package where `xx` is the EDB Postgres Advanced Server version number.

Step2: To launch the worker, it must be registered in the `postgresql.conf` file using the `shared_preload_libraries` parameter, for example:

```
shared_preload_libraries = $libdir/edb_wait_states
```

Step3: Restart the database server. After a successful restart, the background worker begins collecting data.

Step4: To review the data, create the following extension:

```
CREATE EXTENSION edb_wait_states;
```

Step5: To terminate the EDB wait states worker, remove `$libdir/edb_wait_states` from the `shared_preload_libraries` parameter and restart the database server.

The following describes the installation process on a Windows system.

Step1: EDB wait states module is installed with the EDB Modules installer by invoking StackBuilder Plus utility. Follow the onscreen instructions to complete the installation of the EDB Modules .

Step2: To register the worker, modify the `postgresql.conf` file to include the wait states library in the `shared_preload_libraries` configuration parameter. The parameter value must include:

```
shared_preload_libraries = $libdir/edb_wait_states.dll
```

The EDB wait states installation places the `edb_wait_states.dll` library file in the following path:

```
C:\Program Files\edb\as14\lib\
```

Step3: Restart the database server for the changes to take effect. After a successful restart, the background worker gets started and starts collecting the data.

Step4: To view the data, create the following extension:

```
CREATE EXTENSION edb_wait_states;
```

The installer places the `edb_wait_states.control` file in the following path:

```
C:\Program Files\edb\as14\share\extension
```

7.2.2 Terminating the wait states worker

To terminate the EDB wait states worker, use the `DROP EXTENSION` command to drop the `edb_wait_states` extension; then modify the `postgresql.conf` file, removing `$libdir/edb_wait_states.dll` from the `shared_preload_libraries` parameter. Restart the database server after modifying the `postgresql.conf` file to apply your changes.

7.2.3 The wait states interface

The interface includes the functions listed in the following sections. Each of these functions has common input and output parameters. Those parameters are as follows:

- ****start_tsandend_ts(IN).**** Together these specify the time interval and the data_↵
↵within which is to be read. If only `'start_ts'` is specified, the data starting_↵
↵from `'start_ts'` is output. If only `'end_ts'` is provided, data up to `'end_ts'` _↵
↵is output. If none of those are provided, all the data is output. Every function_↵
↵outputs different data. The output of each function is explained below.
- ****query_id(OUT).**** Identifies a normalized query. It **is** internal **hash** code_↵
↵computed **from the** query.
- ****session_id(OUT).**** Identifies a session.
- ****ref_start_tsandref_end_ts(OUT).**** Provide the timestamps of a file containing a_↵
↵particular data point. A data point may be a wait event sample record **or** a query_↵
↵record **or** a session record.

The syntax of each function is given in the following sections.

Note: The examples shown in the following sections are based on the following three queries executed on four different sessions connected to different databases using different users, simultaneously:

```
SELECT schemaname FROM pg_tables, pg_sleep(15) WHERE schemaname <>
pg_catalog; /* ran on 2 sessions */
SELECT tablename FROM pg_tables, pg_sleep(10) WHERE schemaname <>
pg_catalog;
SELECT tablename, schemaname FROM pg_tables, pg_sleep(10) WHERE schemaname
<> pg_catalog;
```

7.2.4 edb_wait_states_data

This function is used to read the data collected by EWSBW.

```
edb_wait_states_data(
  IN start_ts timestamptz default -infinity::timestamptz,
  IN end_ts timestamptz default infinity::timestamptz,
  OUT session_id int4,
  OUT <dbname> text,
  OUT <username> text,
  OUT <query> text,
  OUT <query_start_time> timestamptz,
  OUT <sample_time> timestamptz,
  OUT <wait_event_type> text,
  OUT <wait_event> text
)
```

This function can be used to find out the following:

The queries running in the given duration (defined by `start_ts` and `end_ts`) in all the sessions, and the wait events, if any, they were waiting on. For example:

```
SELECT query, session_id, wait_event_type, wait_event
FROM edb_wait_states_data(start_ts, end_ts);
```

The progress of a session within a given duration (that is, the queries run in a session (`session_id = 100000`) and the wait events the queries waited on). For example:

```
SELECT query, wait_event_type, wait_event
FROM edb_wait_states_data(start_ts, end_ts)
WHERE session_id = 100000;
```

The duration for which the samples are available. For example:

```
SELECT min(sample_time), max(sample_time)
FROM edb_wait_states_data();
```

7.2.5 Parameters

In addition to the common parameters described previously, each row of the output gives the following:

`dbname`

The session's database

`username`

The session's logged in user

`query`

The query running in the session

`query_start_time`

The time when the query started

`sample_time`

The time when wait event data was collected

`wait_event_type`

The type of wait event the session (backend) is waiting on

`wait_event`

The wait event the session (backend) is waiting on

7.2.6 Example

The following is a sample output from the `edb_wait_states_data()` function.

```
edb=# SELECT * FROM edb_wait_states_data();
__OUTPUT__
- [ RECORD 1 ]-----+-----
session_id      | 4398
dbname          | edb
username        | enterprisedb
query           | SELECT schemaname FROM pg_tables, pg_sleep($1) WHERE schemaname <> $2
query_start_time | 17-AUG-18 11:56:05.271962 -04:00
sample_time     | 17-AUG-18 11:56:19.700236 -04:00
wait_event_type  | Timeout
wait_event       | PgSleep
- [ RECORD 2 ]-----+-----
session_id      | 4398
dbname          | edb
username        | enterprisedb
query           | SELECT schemaname FROM pg_tables, pg_sleep($1) WHERE schemaname <> $2
query_start_time | 17-AUG-18 11:56:05.271962 -04:00
sample_time     | 17-AUG-18 11:56:18.699938 -04:00
wait_event_type  | Timeout
wait_event       | PgSleep
- [ RECORD 3 ]-----+-----
```

(continues on next page)

(continued from previous page)

session_id	4398
dbname	edb
username	enterprisedb
query	SELECT schemaname FROM pg_tables, pg_sleep(\$1) WHERE schemaname <> \$2
query_start_time	17-AUG-18 11:56:05.271962 -04:00
sample_time	17-AUG-18 11:56:17.700253 -04:00
wait_event_type	Timeout
wait_event	PgSleep
	.
	.
	.

7.2.7 edb_wait_states_queries

This function gives information about the queries sampled by EWSBW.

```
edb_wait_states_queries(
  IN start_ts timestampz default -infinity::timestampz,
  IN end_ts timestampz default infinity::timestampz,
  OUT query_id int8,
  OUT <query> text,
  OUT ref_start_ts timestampz
  OUT ref_end_ts timestampz
)
```

A new queries file is created periodically and thus, there can be multiple query files generated corresponding to specific intervals.

This function returns all the queries in query files that overlap with the given time interval. A query as shown below, gives all the queries in query files that contained queries sampled between `start_ts` and `end_ts`.

In other words, the function may output queries that did not run in the given interval. To exactly know that the user should use `edb_wait_states_data()`.

```
SELECT query FROM edb_wait_states_queries(start_ts, end_ts);
```

7.2.8 Parameters

In addition to the common parameters described previously, each row of the output gives the following:

query

Normalized query text

7.2.9 Example

The following is a sample output from the `edb_wait_states_queries()` function.

```
edb=# SELECT * FROM edb_wait_states_queries();
__OUTPUT__
- [ RECORD 1 ]+-----
query_id      | 4292540138852956818
query         | SELECT schemaname FROM pg_tables, pg_sleep($1) WHERE schemaname <> $2
ref_start_ts  | 17-AUG-18 11:52:38.698793 -04:00
ref_end_ts    | 18-AUG-18 11:52:38.698793 -04:00
- [ RECORD 2 ]+-----
query_id      | 3754591102365859187
query         | SELECT tablename FROM pg_tables, pg_sleep($1) WHERE schemaname <> $2
ref_start_ts  | 17-AUG-18 11:52:38.698793 -04:00
ref_end_ts    | 18-AUG-18 11:52:38.698793 -04:00
- [ RECORD 3 ]+-----
query_id      | 349089096300352897
query         | SELECT tablename, schemaname FROM pg_tables, pg_sleep($1) WHERE
schemaname <> $2
ref_start_ts  | 17-AUG-18 11:52:38.698793 -04:00
ref_end_ts    | 18-AUG-18 11:52:38.698793 -04:00
```

7.2.10 edb_wait_states_sessions

This function gives information about the sessions sampled by EWSBW.

```
edb_wait_states_sessions(
  IN start_ts timestamptz default -infinity::timestamptz,
  IN end_ts timestamptz default infinity::timestamptz,
  OUT session_id int4,
  OUT <dbname> text,
  OUT <username> text,
  OUT ref_start_ts timestamptz
  OUT ref_end_ts timestamptz
)
```

This function can be used to identify the databases that were connected and/or which users started those sessions. For example:

```
SELECT dbname, username, session_id
FROM edb_wait_states_sessions();
```

Similar to `edb_wait_states_queries()`, this function outputs all the sessions logged in session files that contain sessions sampled within the given interval and not necessarily only the sessions sampled within the given interval. To identify that one should use `edb_wait_states_data()`.

7.2.11 Parameters

In addition to the common parameters described previously, each row of the output gives the following:

`dbname`

The database to which the session is connected

`username`

Login user of the session

7.2.12 Example

The following is a sample output from the `edb_wait_states_sessions()` function.

```
edb=# SELECT * FROM edb_wait_states_sessions();
__OUTPUT__
- [ RECORD 1 ]+-----
session_id    | 4340
dbname        | edb
username      | enterprisedb
ref_start_ts  | 17-AUG-18 11:52:38.698793 -04:00
ref_end_ts    | 18-AUG-18 11:52:38.698793 -04:00
- [ RECORD 2 ]+-----
session_id    | 4398
dbname        | edb
username      | enterprisedb
ref_start_ts  | 17-AUG-18 11:52:38.698793 -04:00
ref_end_ts    | 18-AUG-18 11:52:38.698793 -04:00
- [ RECORD 3 ]+-----
session_id    | 4410
dbname        | db1
username      | user1
ref_start_ts  | 17-AUG-18 11:52:38.698793 -04:00
ref_end_ts    | 18-AUG-18 11:52:38.698793 -04:00
- [ RECORD 4 ]+-----
session_id    | 4422
dbname        | db2
username      | user2
ref_start_ts  | 17-AUG-18 11:52:38.698793 -04:00
ref_end_ts    | 18-AUG-18 11:52:38.698793 -04:00
```

7.2.13 edb_wait_states_samples

This function gives information about wait events sampled by EWSBW.

```
edb_wait_states_samples(
  IN start_ts timestamptz default -infinity::timestamptz,
  IN end_ts timestamptz default infinity::timestamptz,
  OUT query_id int8,
  OUT session_id int4,
  OUT <query_start_time> timestamptz,
```

(continues on next page)

(continued from previous page)

```

OUT <sample_time> timestamptz,
OUT <wait_event_type> text,
OUT <wait_event> text
)

```

Usually, a user would not be required to call this function directly.

7.2.14 Parameters

In addition to the common parameters described previously, each row of the output gives the following:

`query_start_time`

The time when the query started in this session

`sample_time`

The time when wait event data was collected

`wait_event_type`

The type of wait event on which the session is waiting

`wait_event`

The wait event on which the session (backend) is waiting

7.2.15 Example

The following is a sample output from the `edb_wait_states_samples()` function.

```

edb=# SELECT * FROM edb_wait_states_samples();
__OUTPUT__
- [ RECORD 1 ]-----+-----
query_id      | 4292540138852956818
session_id    | 4340
query_start_time | 17-AUG-18 11:56:00.39421 -04:00
sample_time   | 17-AUG-18 11:56:00.699934 -04:00
wait_event_type | Timeout
wait_event     | PgSleep
- [ RECORD 2 ]-----+-----
query_id      | 4292540138852956818
session_id    | 4340
query_start_time | 17-AUG-18 11:56:00.39421 -04:00
sample_time   | 17-AUG-18 11:56:01.699003 -04:00
wait_event_type | Timeout
wait_event     | PgSleep
- [ RECORD 3 ]-----+-----
query_id      | 4292540138852956818
session_id    | 4340
query_start_time | 17-AUG-18 11:56:00.39421 -04:00
sample_time   | 17-AUG-18 11:56:02.70001 -04:00
wait_event_type | Timeout
wait_event     | PgSleep

```

(continues on next page)

(continued from previous page)

```

- [ RECORD 4 ]-----+-----
query_id       | 4292540138852956818
session_id     | 4340
query_start_time | 17-AUG-18 11:56:00.39421 -04:00
sample_time     | 17-AUG-18 11:56:03.700081 -04:00
wait_event_type | Timeout
wait_event      | PgSleep
               | .
               | .
               | .

```

7.2.16 edb_wait_states_purge

The function deletes all the sampled data files (queries, sessions and wait event samples) that were created after `start_ts` and aged (rotated) before `end_ts`.

```

edb_wait_states_purge(
    IN start_ts timestamptz default -infinity::timestamptz,
    IN end_ts timestamptz default infinity::timestamptz
)

```

Usually a user does not need to run this function. The backend should purge those according to the retention age, but in case, that doesn't happen for some reason, this function may be used.

In order to know the duration for which the samples have been retained, use `edb_wait_states_data()` as explained in the previous examples of that function.

7.2.17 Example

The `$PGDATA/edb_wait_states` directory before running `edb_wait_states_purge()`:

```

[root@localhost data]# pwd
/var/lib/edb/as14/data
[root@localhost data]# ls -l edb_wait_states
total 12
-rw----- 1 enterprisedb ... 253 Aug 17 11:56 edb_ws_queries_587836358698793_
  587922758698793
-rw----- 1 enterprisedb ... 1600 Aug 17 11:56 edb_ws_samples_587836358698793_
  587839958698793
-rw----- 1 enterprisedb ... 94 Aug 17 11:56 edb_ws_sessions_587836358698793_
  587922758698793

```

The `$PGDATA/edb_wait_states` directory after running `edb_wait_states_purge()`:

```

edb=# SELECT * FROM edb_wait_states_purge();
__OUTPUT__
edb_wait_states_purge
- -----
(1 row)

```

```
[root@localhost data]# pwd
/var/lib/edb/as14/data
[root@localhost data]# ls -l edb_wait_states
total 0
```


EDB CLONE SCHEMA

- EDB Clone Schema* is an extension module for EDB Postgres Advanced Server that allows you to copy a schema and its database objects from a local or remote database (the source database) to a receiving database (the target database).

The source and target databases can be the same physical database, or different databases within the same database cluster, or separate databases running under different database clusters on separate database server hosts.

Use the following functions with EDB Clone Schema:

- **localcopyschema** This function makes a copy of a schema and its database objects from a source database back into the same database (the target), but with a different schema name than the original. Use this function when the original source schema and the resulting copy are to reside within the same database. See `:ref:localcopyschema<localcopyschema>` for information on the `localcopyschema` function.
- **localcopyschema_nb** This function performs the same purpose as `localcopyschema`, but as a background job, thus freeing up the terminal from which the function was initiated. This is referred to as a *non-blocking* function. See `:ref:localcopyschema_nb<localcopyschema_nb>` for information on the `localcopyschema_nb` function.
- **remotecopyschema** This function makes a copy of a schema and its database objects from a source database to a different target database. Use this function when the original source schema and the resulting copy are to reside in two, separate databases. The separate databases can reside in the same, or in different EDB Postgres Advanced Server database clusters. See `:ref:remotecopyschema<remotecopyschema>` for information on the `remotecopyschema` function.
- **remotecopyschema_nb** This function performs the same purpose as `remotecopyschema`, but as a background job, thus freeing up the terminal from which the function was initiated. This is referred to as a *non-blocking* function. See `:ref:remotecopyschema_nb<remotecopyschema_nb>` for information on the `remotecopyschema_nb` function.
- **process_status_from_log** This function displays the status of the cloning functions. The information is obtained from a log file that must be specified when a cloning function is invoked. See `:ref:process_status_from_log<process_status_from_log>` for information on the `process_status_from_log` function.

- ****remove_log_file_and_job.**** This function deletes the log file created by a [cloning function](#). This function can also be used to delete a job created by the [non-blocking form of the function](#). See [:ref:remove_log_file_and_job<remove_log_file_and_job>](#) for information on the `remove_log_file_and_job` function.

The database objects that can be cloned from one schema to another are the following:

- Data types
- Tables including partitioned tables, excluding foreign tables
- Indexes
- Constraints
- Sequences
- View definitions
- Materialized views
- Private synonyms
- Table triggers, but excluding event triggers
- Rules
- Functions
- Procedures
- Packages
- Comments for all supported object types
- Access control lists (ACLs) for all supported object types

The following database objects cannot be cloned:

- Large objects (Postgres LOBs and BFILEs)
- Logical replication attributes for a table
- Database links
- Foreign data wrappers
- Foreign tables
- Event triggers
- Extensions (For cloning objects that rely on extensions, see the third bullet point in the following limitations list.)
- Row level security
- Policies
- Operator class

In addition, the following limitations apply:

- EDB Clone Schema is supported on EDB Postgres Advanced Server only when a dialect of `Compatible with Oracle` is specified on the EDB Postgres Advanced Server Dialect dialog during installation, or when the `--redwood-like` keywords are included during a text mode installation or cluster initialization.
- The source code within functions, procedures, triggers, packages, etc., are not modified after being copied to the target schema. If such programs contain coded references to objects with schema names, the programs may fail upon invocation in the target schema if such schema names are no longer consistent within the target schema.

- Cross schema object dependencies are not resolved. If an object in the target schema depends upon an object in another schema, this dependency is not resolved by the cloning functions.
- For remote cloning, if an object in the source schema is dependent upon an extension, then this extension must be created in the public schema of the remote database before invoking the remote cloning function.
- At most, 16 copy jobs can run in parallel to clone schemas, whereas each job can have at most 16 worker processes to copy table data in parallel.
- Queries being run by background workers cannot be cancelled.

The following section describes how to set up EDB Clone Schema on the databases.

8.1 Setup process

Several extensions along with the PL/Perl language must be installed on any database to be used as the source or target database by an EDB Clone Schema function.

In addition, some configuration parameters in the `postgresql.conf` file of the database servers may benefit from some modification.

The following is the setup instructions for these requirements.

8.1.1 Installing extensions and PL/Perl

The following describes the steps to install the required extensions and the PL/Perl language.

These steps must be performed on any database to be used as the source or target database by an EDB Clone Schema function.

Step1: The following extensions must be installed on the database:

- `postgres_fdw`
- `dblink`
- `adminpack`
- `pgagent`

Ensure that `pgAgent` is installed before creating the `pgagent` extension. On Linux, you can use the `edb-asxx-pgagent` RPM package where `xx` is the EDB Postgres Advanced Server version number to install `pgAgent`. On Windows, use StackBuilder Plus to download and install `pgAgent`.

The previously listed extensions can be installed by the following commands if they don't already exist:

```
CREATE EXTENSION postgres_fdw SCHEMA public;
CREATE EXTENSION dblink SCHEMA public;
CREATE EXTENSION adminpack;
CREATE EXTENSION pgagent;
```

For more information about using the `CREATE EXTENSION` command, see the PostgreSQL core documentation at:

<https://www.postgresql.org/docs/current/static/sql-createextension.html>

Step2: Modify the `postgresql.conf` file.

Modify the `postgresql.conf` file by adding `$libdir/parallel_clone` to the `shared_preload_libraries` configuration parameter as shown by the following example:

```
shared_preload_libraries = $libdir/dbms_pipe,$libdir/dbms_aq,$libdir/parallel_clone
```

Step3: The Perl Procedural Language (PL/Perl) must be installed on the database and the `CREATE TRUSTED LANGUAGE plperl` command must be run. For Linux, install PL/Perl using the `edb-asxx-server-plperl` RPM package where `xx` is the EDB Postgres Advanced Server version number. For Windows, use the EDB Postgres Language Pack. For information on EDB Language Pack, see the *EDB Postgres Language Pack Guide* available at:

<https://www.enterprisedb.com/docs>

Step4: Connect to the database as a superuser and run the following command:

```
CREATE TRUSTED LANGUAGE plperl;
```

For more information about using the `CREATE LANGUAGE` command, see the PostgreSQL core documentation at:

<https://www.postgresql.org/docs/current/static/sql-createlanguage.html>

8.1.2 Setting configuration parameters

The following sections describe configuration parameters that may need to be altered in the `postgresql.conf` file.

8.1.3 Performance configuration parameters

You may need to tune the system for copying a large schema as part of one transaction.

Tuning of configuration parameters is for the source database server referenced in a cloning function.

The configuration parameters in the `postgresql.conf` file that may need to be tuned include the following:

- `**work_mem.**` Specifies the amount of memory to be used by internal sort operations and hash tables before writing to temporary disk files.
- `**maintenance_work_mem.**` Specifies the maximum amount of memory to be used by maintenance operations, such as ``VACUUM``, ``CREATE INDEX``, and ``ALTER TABLE ADD FOREIGN KEY``.
- `**max_worker_processes.**` Sets the maximum number of background processes that the system can support.
- `**checkpoint_timeout.**` Maximum time between automatic WAL checkpoints, in seconds.
- `**checkpoint_completion_target.**` Specifies the target of checkpoint completion, as a fraction of total time between checkpoints.
- `**checkpoint_flush_after.**` Whenever more than ``checkpoint_flush_after`` bytes have been written while performing a checkpoint, attempt to force the OS to issue these writes to the underlying storage.
- `**max_wal_size.**` Maximum size to let the WAL grow to between automatic WAL checkpoints.

- ****max_locks_per_transaction.**** This parameter controls the average number of `object` locks allocated **for** each transaction; individual transactions can lock **more** objects **as long as** the locks of **all** transactions fit **in** the lock table.

For information about the configuration parameters, see the PostgreSQL core documentation at:

<https://www.postgresql.org/docs/current/static/runtime-config.html>

8.1.4 Status logging

Status logging by the cloning functions creates log files in the directory specified by the `log_directory` parameter in the `postgresql.conf` file for the database server to which you are connected when invoking the cloning function.

The default location is `PGDATA/log` as shown by the following:

```
# log_directory = log           # directory where log files are written,
                                # can be absolute or relative to PGDATA
```

This directory must exist prior to running a cloning function.

The name of the log file is determined by what you specify in the parameter list when invoking the cloning function.

To display the status from a log file, use the `process_status_from_log` function.

To delete a log file, use the `remove_log_file_and_job` function, or simply navigate to the log directory and delete it manually.

8.1.5 Installing EDB Clone Schema

The following are the directions for installing EDB Clone Schema.

These steps must be performed on any database to be used as the source or target database by an EDB Clone Schema function.

Step1: If you had previously installed an older version of the `edb_cloneschema` extension, then you must run the following command:

```
DROP EXTENSION parallel_clone CASCADE;
```

This command also drops the `edb_cloneschema` extension.

Step2: Install the extensions using the following commands:

```
CREATE EXTENSION parallel_clone SCHEMA public;

CREATE EXTENSION edb_cloneschema;
```

Make sure you create the `parallel_clone` extension before creating the `edb_cloneschema` extension.

8.1.6 Creating the foreign servers and user mappings

When using one of the local cloning functions, `localcopyschema` or `localcopyschema_nb`, one of the required parameters includes a single, foreign server for identifying the database server along with its database that is the source and the receiver of the cloned schema.

When using one of the remote cloning functions, `remotecopyschema` or `remotecopyschema_nb`, two of the required parameters include two foreign servers. The foreign server specified as the first parameter identifies the source database server along with its database that is the provider of the cloned schema. The foreign server specified as the second parameter identifies the target database server along with its database that is the receiver of the cloned schema.

For each foreign server, a user mapping must be created. When a selected database superuser invokes a cloning function, that database superuser who invokes the function must have been mapped to a database user name and password that has access to the foreign server that is specified as a parameter in the cloning function.

For general information about foreign data, foreign servers, and user mappings, see the PostgreSQL core documentation at:

<https://www.postgresql.org/docs/current/static/ddl-foreign-data.html>

The following two sections describe how these foreign servers and user mappings are defined.

8.1.7 Foreign server and user mapping for local cloning runctions

For the `localcopyschema` and `localcopyschema_nb` functions, the source and target schemas are both within the same database of the same database server. Thus, only one foreign server must be defined and specified for these functions. This foreign server is also referred to as the *local server*.

This server is referred to as the local server because this server is the one to which you must be connected when invoking the `localcopyschema` or `localcopyschema_nb` function.

The user mapping defines the connection and authentication information for the foreign server.

This foreign server and user mapping must be created within the database of the local server in which the cloning is to occur.

The database user for whom the user mapping is defined must be a superuser and the user connected to the local server when invoking an EDB Clone Schema function.

The following example creates the foreign server for the database containing the schema to be cloned, and to receive the cloned schema as well.

```
CREATE SERVER local_server FOREIGN DATA WRAPPER postgres_fdw
  OPTIONS(
    host localhost,
    port 5444,
    dbname edb
  );
```

For more information about using the `CREATE SERVER` command, see the PostgreSQL core documentation at:

<https://www.postgresql.org/docs/current/static/sql-createserver.html>

The user mapping for this server is the following:

```
CREATE USER MAPPING FOR enterprisedb SERVER local_server
  OPTIONS (
    user enterprisedb,
```

(continues on next page)

(continued from previous page)

```
password password
);
```

For more information about using the `CREATE USER MAPPING` command, see the PostgreSQL core documentation at: <https://www.postgresql.org/docs/current/sql-createusermapping.html>

The following `psql` commands show the foreign server and user mapping:

```
edb=# \des+
__OUTPUT__
List of foreign servers
- [ RECORD 1 ]-----+-----
Name              | local_server
Owner              | enterprisedb
Foreign-data wrapper | postgres_fdw
Access privileges  |
Type               |
Version            |
FDW options        | (host localhost, port 5444, dbname edb)
Description        |
```

```
edb=# \deu+
__OUTPUT__
List of user mappings
Server | User name | FDW options
-----+-----+-----
local_server | enterprisedb | ("user" enterprisedb, password password)
(1 row)
```

When database superuser `enterprisedb` invokes a cloning function, the database user `enterprisedb` with its password is used to connect to `local_server` on the `localhost` with port 5444 to database `edb`.

In this case, the mapped database user, `enterprisedb`, and the database user, `enterprisedb`, used to connect to the local `edb` database happen to be the same, identical database user, but that is not an absolute requirement.

For specific usage of these foreign server and user mapping examples, see the example given in `localcopyschema`.

8.1.8 Foreign server and user mapping for remote cloning functions

For the `remotecopyschema` and `remotecopyschema_nb` functions, the source and target schemas are in different databases of either the same or different database servers. Thus, two foreign servers must be defined and specified for these functions.

The foreign server defining the originating database server and its database containing the source schema to be cloned is referred to as the *source server* or the *remote server*.

The foreign server defining the database server and its database to receive the schema to be cloned is referred to as the *target server* or the *local server*.

The target server is also referred to as the local server because this server is the one to which you must be connected when invoking the `remotecopyschema` or `remotecopyschema_nb` function.

The user mappings define the connection and authentication information for the foreign servers.

All of these foreign servers and user mappings must be created within the target database of the target/local server.

The database user for whom the user mappings are defined must be a superuser and the user connected to the local server when invoking an EDB Clone Schema function.

The following example creates the foreign server for the local, target database that is to receive the cloned schema.

```
CREATE SERVER tgt_server FOREIGN DATA WRAPPER postgres_fdw
OPTIONS(
  host localhost,
  port 5444,
  dbname tgtdb
);
```

The user mapping for this server is the following:

```
CREATE USER MAPPING FOR enterprisedb SERVER tgt_server
OPTIONS (
  user tgtuser,
  password tgtpassword
);
```

The following example creates the foreign server for the remote, source database that is to be the source for the cloned schema.

```
CREATE SERVER src_server FOREIGN DATA WRAPPER postgres_fdw
OPTIONS(
  host 192.168.2.28,
  port 5444,
  dbname srcdb
);
```

The user mapping for this server is the following:

```
CREATE USER MAPPING FOR enterprisedb SERVER src_server
OPTIONS (
  user srcuser,
  password srcpassword
);
```

The following psql commands show the foreign servers and user mappings:

```
tgtdb=# \des+
__OUTPUT__
List of foreign servers
- [ RECORD 1 ]-----+-----
Name           | src_server
Owner          | tgtuser
Foreign-data wrapper | postgres_fdw
Access privileges |
Type           |
Version        |
FDW options    | (host 192.168.2.28, port 5444, dbname srcdb)
Description    |
- [ RECORD 2 ]-----+-----
Name           | tgt_server
Owner          | tgtuser
```

(continues on next page)

(continued from previous page)

Foreign-data wrapper	postgres_fdw
Access privileges	
Type	
Version	
FDW options	(host localhost, port 5444, dbname tgtdb)
Description	

tgtdb=# \deu+

__OUTPUT__

List of user mappings		
Server	User name	FDW options
-----+-----+-----		
src_server	enterprisedb	("user" srcuser, password srcpassword)
tgt_server	enterprisedb	("user" tgtuser, password tgtpassword)

(2 rows)

When database superuser `enterprisedb` invokes a cloning function, the database user `tgtuser` with password `tgtpassword` is used to connect to `tgt_server` on the localhost with port 5444 to database `tgtdb`.

In addition, database user `srcuser` with password `srcpassword` connects to `src_server` on host `192.168.2.28` with port 5444 to database `srcdb`.

Note: Be sure the `pg_hba.conf` file of the database server running the source database `srcdb` has an appropriate entry permitting connection from the target server location (address `192.168.2.27` in the following example) connecting with the database user `srcuser` that was included in the user mapping for the foreign server `src_server` defining the source server and database.

#	TYPE	DATABASE	USER	ADDRESS	METHOD
# "local" is for Unix domain socket connections only					
local	all		all		md5
# IPv4 local connections:					
host	srcdb		srcuser	192.168.2.27/32	md5

For specific usage of these foreign server and user mapping examples, see the example given `remotecopyschema`.

8.2 EDB Clone Schema functions

The EDB Clone Schema functions are created in the `edb_util` schema when the `parallel_clone` and `edb_cloneschema` extensions are installed.

Verify the following conditions before using an EDB Clone Schema function:

- You are connected to the target or local database as the database superuser defined in the `CREATE USER MAPPING` command for the foreign server of the target or local database.
- The `edb_util` schema is in the search path, or the cloning function is to be invoked with the `edb_util` prefix.
- The target schema does not exist in the target database.

- When using the remote copy functions, if the `on_tblspace` parameter is to be set to `true` , then the target database cluster contains all tablespaces that are referenced by objects in the source schema, otherwise creation of the DDL statements for those database objects fails in the target schema. This causes a failure of the cloning process.
- When using the remote copy functions, if the `copy_acls` parameter is to be set to `true` , then all roles that have `GRANT` privileges on objects in the source schema exist in the target database cluster, otherwise granting of privileges to those roles fails in the target schema. This causes a failure of the cloning process.
- `pgAgent` is running against the target database if the non-blocking form of the function is to be used.

For information about `pgAgent`, see the `pgAdmin` documentation available at:

<https://www.pgadmin.org/docs/pgadmin4/dev/pgagent.html>

Note that `pgAgent` is provided as a component with EDB Postgres Advanced Server.

8.2.1 `localcopyschema`

The `localcopyschema` function copies a schema and its database objects within a local database specified within the `source_fdw` foreign server from the source schema to the specified target schema within the same database.

```
localcopyschema(  
  <source_fdw> TEXT,  
  <source_schema> TEXT,  
  <target_schema> TEXT,  
  <log_filename> TEXT  
  [, <on_tblspace> BOOLEAN  
  [, <verbose_on> BOOLEAN  
  [, <copy_acls> BOOLEAN  
  [, <worker_count> INTEGER ]]]  
)
```

A `BOOLEAN` value is returned by the function. If the function succeeds, then `true` is returned. If the function fails, then `false` is returned.

The `source_fdw`, `source_schema`, `target_schema`, and `log_filename` are required parameters while all other parameters are optional.

8.2.2 Parameters

`source_fdw`

Name of the foreign server managed by the `postgres_fdw` foreign data wrapper from which database objects are to be cloned.

`source_schema`

Name of the schema from which database objects are to be cloned.

`target_schema`

Name of the schema into which database objects are to be cloned from the source schema.

`log_filename`

Name of the log file in which information from the function is recorded. The log file is created under the directory specified by the `log_directory` configuration parameter in the `postgresql.conf` file.

on_tblspace

BOOLEAN value to specify whether or not database objects are to be created within their tablespaces. If `false` is specified, then the `TABLESPACE` clause is not included in the applicable CREATE DDL statement when added to the target schema. If `true` is specified, then the `TABLESPACE` clause is included in the CREATE DDL statement when added to the target schema. If the `on_tblspace` parameter is omitted, the default value is `false`.

verbose_on

BOOLEAN value to specify whether or not the DDLs are to be printed in `log_filename` when creating objects in the target schema. If `false` is specified, then DDLs are not printed. If `true` is specified, then DDLs are printed. If omitted, the default value is `false`.

copy_acls

BOOLEAN value to specify whether or not the access control list (ACL) is to be included while creating objects in the target schema. The access control list is the set of `GRANT` privilege statements. If `false` is specified, then the access control list is not included for the target schema. If `true` is specified, then the access control list is included for the target schema. If the `copy_acls` parameter is omitted, the default value is `false`.

worker_count

Number of background workers to perform the clone in parallel. If omitted, the default value is 1.

8.2.3 Example

The following example shows the cloning of schema `edb` containing a set of database objects to target schema `edbcopy`, both within database `edb` as defined by `local_server`.

The example is for the following environment:

- Host on which the database server is running: `localhost`
- Port of the database server: `5444`
- Database source/target of the clone: `edb`
- Foreign server (`local_server`) and user mapping with the information of the preceding bullet points
- Source schema: `edb`
- Target schema: `edbcopy`
- Database superuser to invoke `localcopyschema`: `enterprisedb`

Before invoking the function, the connection is made by database user `enterprisedb` to database `edb`.

```
edb=# SET search_path TO "$user",public,edb_util;
SET
edb=# SHOW search_path;
__OUTPUT__
      search_path
-----
"$user", public, edb_util
(1 row)
```

```
edb=# SELECT localcopyschema (local_server,edb,edbcopy,clone_edb_edbcopy);
__OUTPUT__
 localcopyschema
-----
```

(continues on next page)

(continued from previous page)

```
t
(1 row)
```

The following displays the logging status using the `process_status_from_log` function:

```
edb=# SELECT process_status_from_log(clone_edb_edbcopy);
__OUTPUT__

              process_status_from_log
-----
(FINISH,"2017-06-29 11:07:36.830783-04",3855,INFO,"STAGE: FINAL","successfully cloned_
schema")
(1 row)
```

After the clone has completed, the following shows some of the database objects copied to the `edbcopy` schema:

```
edb=# SET search_path TO edbcopy;
SET
edb=# \dt+
__OUTPUT__

              List of relations
 Schema | Name   | Type  | Owner      | Size      | Description
-----+-----+-----+-----+-----+-----
 edbcopy | dept   | table | enterprisedb | 8192 bytes |
 edbcopy | emp    | table | enterprisedb | 8192 bytes |
 edbcopy | jobhist | table | enterprisedb | 8192 bytes |
(3 rows)
```

```
edb=# \dv
__OUTPUT__

              List of relations
 Schema | Name    | Type  | Owner      |
-----+-----+-----+-----+
 edbcopy | salesemp | view  | enterprisedb |
(1 row)
```

```
edb=# \di
__OUTPUT__

              List of relations
 Schema | Name           | Type  | Owner      | Table
-----+-----+-----+-----+-----+
 edbcopy | dept_dname_uq  | index | enterprisedb | dept
 edbcopy | dept_pk        | index | enterprisedb | dept
 edbcopy | emp_pk         | index | enterprisedb | emp
 edbcopy | jobhist_pk     | index | enterprisedb | jobhist
(4 rows)
```

```
edb=# \ds
__OUTPUT__

              List of relations
 Schema | Name   | Type  | Owner
-----+-----+-----+-----
```

(continues on next page)

(continued from previous page)

```
edbcopy | next_empno | sequence | enterprisedb
(1 row)
```

```
edb=# SELECT DISTINCT schema_name, name, type FROM user_source WHERE
schema_name = EDBCOPY ORDER BY type, name;
```

```
__OUTPUT__
```

schema_name	name	type
EDBCOPY	EMP_COMP	FUNCTION
EDBCOPY	HIRE_CLERK	FUNCTION
EDBCOPY	HIRE_SALESMAN	FUNCTION
EDBCOPY	NEW_EMPNO	FUNCTION
EDBCOPY	EMP_ADMIN	PACKAGE
EDBCOPY	EMP_ADMIN	PACKAGE BODY
EDBCOPY	EMP_QUERY	PROCEDURE
EDBCOPY	EMP_QUERY_CALLER	PROCEDURE
EDBCOPY	LIST_EMP	PROCEDURE
EDBCOPY	SELECT_EMP	PROCEDURE
EDBCOPY	EMP_SAL_TRIG	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_a_19991"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_a_19992"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_a_19999"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_a_20000"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_a_20004"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_a_20005"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_c_19993"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_c_19994"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_c_20001"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_c_20002"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_c_20006"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_c_20007"	TRIGGER
EDBCOPY	USER_AUDIT_TRIG	TRIGGER

```
(24 rows)
```

8.2.4 localcopyschema_nb

The `localcopyschema_nb` function copies a schema and its database objects within a local database specified within the `source_fdw` foreign server from the source schema to the specified target schema within the same database, but in a non-blocking manner as a job submitted to pgAgent.

```
localcopyschema_nb(
  <source_fdw> TEXT,
  <source> TEXT,
  <target> TEXT,
  <log_filename> TEXT
  [, <on_tblspace> BOOLEAN
  [, <verbose_on> BOOLEAN
  [, <copy_acls> BOOLEAN
  [, <worker_count> INTEGER ]]]
)
```

An INTEGER value job ID is returned by the function for the job submitted to pgAgent. If the function fails, then null is returned.

The `source_fdw`, `source`, `target`, and `log_filename` are required parameters while all other parameters are optional.

After completion of the pgAgent job, remove the job with the `remove_log_file_and_job` function.

8.2.5 Parameters

`source_fdw`

Name of the foreign server managed by the `postgres_fdw` foreign data wrapper from which database objects are to be cloned.

`source`

Name of the schema from which database objects are to be cloned.

`target`

Name of the schema into which database objects are to be cloned from the source schema.

`log_filename`

Name of the log file in which information from the function is recorded. The log file is created under the directory specified by the `log_directory` configuration parameter in the `postgresql.conf` file.

`on_tablespace`

BOOLEAN value to specify whether or not database objects are to be created within their tablespaces. If `false` is specified, then the `TABLESPACE` clause is not included in the applicable `CREATE DDL` statement when added to the target schema. If `true` is specified, then the `TABLESPACE` clause is included in the `CREATE DDL` statement when added to the target schema. If the `on_tablespace` parameter is omitted, the default value is `false`.

`verbose_on`

BOOLEAN value to specify whether or not the DDLs are to be printed in `log_filename` when creating objects in the target schema. If `false` is specified, then DDLs are not printed. If `true` is specified, then DDLs are printed. If omitted, the default value is `false`.

`copy_acls`

BOOLEAN value to specify whether or not the access control list (ACL) is to be included while creating objects in the target schema. The access control list is the set of `GRANT` privilege statements. If `false` is specified, then the access control list is not included for the target schema. If `true` is specified, then the access control list is included for the target schema. If the `copy_acls` parameter is omitted, the default value is `false`.

`worker_count`

Number of background workers to perform the clone in parallel. If omitted, the default value is 1.

8.2.6 Example

The same cloning operation is performed as the example in `localcopyschema` , but using the non-blocking function `localcopyschema_nb` .

The following command can be used to observe if pgAgent is running on the appropriate local database:

```
[root@localhost ~]# ps -ef | grep pgagent
root      4518      1  0 11:35 pts/1    00:00:00 pgagent -s /tmp/pgagent_edb_log
hostaddr=127.0.0.1 port=5444 dbname=edb user=enterprisedb password=password
root      4525    4399  0 11:35 pts/1    00:00:00 grep --color=auto pgagent
```

If pgAgent is not running, it can be started as shown by the following. The pgagent program file is located in the bin subdirectory of the EDB Postgres Advanced Server installation directory.

```
[root@localhost bin]# ./pgagent -l 2 -s /tmp/pgagent_edb_log hostaddr=127.0.0.1 port=5444
dbname=edb user=enterprisedb password=password
```

Note: The `pgagent -l 2` option starts pgAgent in *DEBUG* mode, which logs continuous debugging information into the log file specified with the `-s` option. Use a lower value for the `-l` option, or omit it entirely to record less information.

The `localcopyschema_nb` function returns the job ID shown as 4 in the example.

```
edb=# SELECT edb_util.localcopyschema_nb (local_server,edb,edbcopy,clone_edb_edbcopy);
__OUTPUT__
localcopyschema_nb
-----
                4
(1 row)
```

The following displays the job status:

```
edb=# SELECT edb_util.process_status_from_log(clone_edb_edbcopy);
__OUTPUT__
process_status_from_log
-----
(FINISH,"29-JUN-17 11:39:11.620093 -04:00",4618,INFO,"STAGE: FINAL","successfully_
cloned schema")
(1 row)
```

The following removes the pgAgent job:

```
edb=# SELECT edb_util.remove_log_file_and_job (4);
__OUTPUT__
remove_log_file_and_job
-----
t
(1 row)
```

8.2.7 remotecopyschema

The `remotecopyschema` function copies a schema and its database objects from a source schema in the remote source database specified within the `source_fdw` foreign server to a target schema in the local target database specified within the `target_fdw` foreign server.

```
remotecopyschema(  
  <source_fdw> TEXT,  
  <target_fdw> TEXT,  
  <source_schema> TEXT,  
  <target_schema> TEXT,  
  <log_filename> TEXT  
  [, <on_tblspace> BOOLEAN  
  [, <verbose_on> BOOLEAN  
  [, <copy_acls> BOOLEAN  
  [, <worker_count> INTEGER ]]]  
)
```

A `BOOLEAN` value is returned by the function. If the function succeeds, then `true` is returned. If the function fails, then `false` is returned.

The `source_fdw`, `target_fdw`, `source_schema`, `target_schema`, and `log_filename` are required parameters while all other parameters are optional.

8.2.8 Parameters

source_fdw

Name of the foreign server managed by the `postgres_fdw` foreign data wrapper from which database objects are to be cloned.

target_fdw

Name of the foreign server managed by the `postgres_fdw` foreign data wrapper to which database objects are to be cloned.

source_schema

Name of the schema from which database objects are to be cloned.

target_schema

Name of the schema into which database objects are to be cloned from the source schema.

log_filename

Name of the log file in which information from the function is recorded. The log file is created under the directory specified by the `log_directory` configuration parameter in the `postgresql.conf` file.

on_tblspace

`BOOLEAN` value to specify whether or not database objects are to be created within their tablespaces. If `false` is specified, then the `TABLESPACE` clause is not included in the applicable `CREATE DDL` statement when added to the target schema. If `true` is specified, then the `TABLESPACE` clause is included in the `CREATE DDL` statement when added to the target schema. If the `on_tblspace` parameter is omitted, the default value is `false`.

Note: If `true` is specified and a database object has a `TABLESPACE` clause, but that tablespace does not exist in the target database cluster, then the cloning function fails.

verbose_on

BOOLEAN value to specify whether or not the DDLs are to be printed in `log_filename` when creating objects in the target schema. If `false` is specified, then DDLs are not printed. If `true` is specified, then DDLs are printed. If omitted, the default value is `false`.

copy_acls

BOOLEAN value to specify whether or not the access control list (ACL) is to be included while creating objects in the target schema. The access control list is the set of `GRANT` privilege statements. If `false` is specified, then the access control list is not included for the target schema. If `true` is specified, then the access control list is included for the target schema. If the `copy_acls` parameter is omitted, the default value is `false`.

Note: If `true` is specified and a role with `GRANT` privilege does not exist in the target database cluster, then the cloning function fails.

worker_count

Number of background workers to perform the clone in parallel. If omitted, the default value is 1.

8.2.9 Example

The following example shows the cloning of schema `srcschema` within database `srcdb` as defined by `src_server` to target schema `tgtschema` within database `tgtdb` as defined by `tgt_server`.

The source server environment:

- Host on which the source database server is running: `192.168.2.28`
- Port of the source database server: `5444`
- Database source of the clone: `srcdb`
- Foreign server (`src_server`) and user mapping with the information of the preceding bullet points
- Source schema: `srcschema`

The target server environment:

- Host on which the target database server is running: `localhost`
- Port of the target database server: `5444`
- Database target of the clone: `tgtdb`
- Foreign server (`tgt_server`) and user mapping with the information of the preceding bullet points
- Target schema: `tgtschema`
- Database superuser to invoke `remotecopyschema`: `enterprisedb`

Before invoking the function, the connection is made by database user `enterprisedb` to database `tgtdb`. A `worker_count` of 4 is specified for this function.

```
tgtdb=# SELECT edb_util.remotecopyschema
(src_server,tgt_server,srcschema,tgtschema,clone_rmt_src_tgt,worker_count => 4);
__OUTPUT__
remotecopyschema
- - - - -
t
(1 row)
```

The following displays the status from the log file during various points in the cloning process:

```

tgtodb=# SELECT edb_util.process_status_from_log(clone_rmt_src_tgt);
__OUTPUT__
                                     process_status_from_log
-----
(RUNNING,"28-JUN-17 13:18:05.299953 -04:00",4021,INFO,"STAGE: DATA-COPY","[0][0]
↳successfully
copied data in [tgtschema.pgbench_tellers]
")
(1 row)

```

```

tgtodb=# SELECT edb_util.process_status_from_log(clone_rmt_src_tgt);
__OUTPUT__
                                     process_status_from_log
-----
(RUNNING,"28-JUN-17 13:18:06.634364 -04:00",4022,INFO,"STAGE: DATA-COPY","[0][1]
↳successfully
copied data in [tgtschema.pgbench_history]
")
(1 row)

```

```

tgtodb=# SELECT edb_util.process_status_from_log(clone_rmt_src_tgt);
__OUTPUT__
                                     process_status_from_log
-----
(RUNNING,"28-JUN-17 13:18:10.550393 -04:00",4039,INFO,"STAGE: POST-DATA","CREATE
↳PRIMARY KEY
CONSTRAINT pgbench_tellers_pkey successful"
)
(1 row)

```

```

tgtodb=# SELECT edb_util.process_status_from_log(clone_rmt_src_tgt);
__OUTPUT__
                                     process_status_from_log
-----
(FINISH,"28-JUN-17 13:18:12.019627 -04:00",4039,INFO,"STAGE: FINAL","successfully clone
schema into tgtschema")
(1 row)

```

The following shows the cloned tables:

```
tgtdb=# \dt+
__OUTPUT__
```

List of relations					
Schema	Name	Type	Owner	Size	Description
tgtschema	pgbench_accounts	table	enterprisedb	256 MB	
tgtschema	pgbench_branches	table	enterprisedb	8192 bytes	
tgtschema	pgbench_history	table	enterprisedb	25 MB	
tgtschema	pgbench_tellers	table	enterprisedb	16 kB	

(4 rows)

When the remotecopyschema function was invoked, four background workers were specified.

The following portion of the log file clone_rmt_src_tgt shows the status of the parallel data copying operation using four background workers:

```
Wed Jun 28 13:18:05.232949 2017 EDT: [4019] INFO: [STAGE: DATA-COPY] [0] table count [4]
Wed Jun 28 13:18:05.233321 2017 EDT: [4019] INFO: [STAGE: DATA-COPY] [0][0] worker_
↳started to
copy data
Wed Jun 28 13:18:05.233640 2017 EDT: [4019] INFO: [STAGE: DATA-COPY] [0][1] worker_
↳started to
copy data
Wed Jun 28 13:18:05.233919 2017 EDT: [4019] INFO: [STAGE: DATA-COPY] [0][2] worker_
↳started to
copy data
Wed Jun 28 13:18:05.234231 2017 EDT: [4019] INFO: [STAGE: DATA-COPY] [0][3] worker_
↳started to
copy data
Wed Jun 28 13:18:05.298174 2017 EDT: [4024] INFO: [STAGE: DATA-COPY] [0][3] successfully
copied data in [tgtschema.pgbench_branches]
Wed Jun 28 13:18:05.299913 2017 EDT: [4021] INFO: [STAGE: DATA-COPY] [0][0] successfully
copied data in [tgtschema.pgbench_tellers]
Wed Jun 28 13:18:06.634310 2017 EDT: [4022] INFO: [STAGE: DATA-COPY] [0][1] successfully
copied data in [tgtschema.pgbench_history]
Wed Jun 28 13:18:10.477333 2017 EDT: [4023] INFO: [STAGE: DATA-COPY] [0][2] successfully
copied data in [tgtschema.pgbench_accounts]
Wed Jun 28 13:18:10.477609 2017 EDT: [4019] INFO: [STAGE: DATA-COPY] [0] all workers_
↳finished
[4]
Wed Jun 28 13:18:10.477654 2017 EDT: [4019] INFO: [STAGE: DATA-COPY] [0] copy done [4]_
↳tables
Wed Jun 28 13:18:10.493938 2017 EDT: [4019] INFO: [STAGE: DATA-COPY] successfully copied_
↳data
into tgtschema
```

Note that the DATA-COPY log message includes two, square bracket numbers (for example, [0][3]).

The first number is the job index whereas the second number is the worker index. The worker index values range from 0 to 3 for the four background workers.

In case two clone schema jobs are running in parallel, the first log file has 0 as the job index whereas the second has 1 as the job index.

8.2.10 remotecopyschema_nb

The `remotecopyschema_nb` function copies a schema and its database objects from a source schema in the remote source database specified within the `source_fdw` foreign server to a target schema in the local target database specified within the `target_fdw` foreign server, but in a non-blocking manner as a job submitted to pgAgent.

```
remotecopyschema_nb(  
  <source_fdw> TEXT,  
  <target_fdw> TEXT,  
  <source> TEXT,  
  <target> TEXT,  
  <log_filename> TEXT  
  [, <on_tblspace> BOOLEAN  
  [, <verbose_on> BOOLEAN  
  [, <copy_acls> BOOLEAN  
  [, <worker_count> INTEGER ]]]  
)
```

An `INTEGER` value job ID is returned by the function for the job submitted to pgAgent. If the function fails, then null is returned.

The `source_fdw`, `target_fdw`, `source`, `target`, and `log_filename` are required parameters while all other parameters are optional.

After completion of the pgAgent job, remove the job with the `remove_log_file_and_job` function.

8.2.11 Parameters

`source_fdw`

Name of the foreign server managed by the `postgres_fdw` foreign data wrapper from which database objects are to be cloned.

`target_fdw`

Name of the foreign server managed by the `postgres_fdw` foreign data wrapper to which database objects are to be cloned.

`source`

Name of the schema from which database objects are to be cloned.

`target`

Name of the schema into which database objects are to be cloned from the source schema.

`log_filename`

Name of the log file in which information from the function is recorded. The log file is created under the directory specified by the `log_directory` configuration parameter in the `postgresql.conf` file.

`on_tblspace`

`BOOLEAN` value to specify whether or not database objects are to be created within their tablespaces. If `false` is specified, then the `TABLESPACE` clause is not included in the applicable `CREATE DDL` statement when added to the target schema. If `true` is specified, then the `TABLESPACE` clause is included in the `CREATE DDL` statement when added to the target schema. If the `on_tblspace` parameter is omitted, the default value is `false`.

Note: If *true* is specified and a database object has a *TABLESPACE* clause, but that tablespace does not exist in the target database cluster, then the cloning function fails.

verbose_on

BOOLEAN value to specify whether or not the DDLs are to be printed in *log_filename* when creating objects in the target schema. If *false* is specified, then DDLs are not printed. If *true* is specified, then DDLs are printed. If omitted, the default value is *false*.

copy_acls

BOOLEAN value to specify whether or not the access control list (ACL) is to be included while creating objects in the target schema. The access control list is the set of GRANT privilege statements. If *false* is specified, then the access control list is not included for the target schema. If *true* is specified, then the access control list is included for the target schema. If the *copy_acls* parameter is omitted, the default value is *false*. **Note:** If *true* is specified and a role with GRANT privilege does not exist in the target database cluster, then the cloning function fails.

worker_count

Number of background workers to perform the clone in parallel. If omitted, the default value is 1.

8.2.12 Example

The same cloning operation is performed as the example in *remotecopyschema*, but using the non-blocking function *remotecopyschema_nb*.

The following command starts pgAgent on the target database *tgtdb*. The *pgagent* program file is located in the *bin* subdirectory of the EDB Postgres Advanced Server installation directory.

```
[root@localhost bin]# ./pgagent -l 1 -s /tmp/pgagent_tgtdb_log hostaddr=127.0.0.1
↪port=5444
user=enterprisedb dbname=tgtdb password=password
```

The *remotecopyschema_nb* function returns the job ID shown as 2 in the example.

```
tgtdb=# SELECT edb_util.remotecopyschema_nb
(src_server,tgt_server,srcschema,tgtschema,clone_rmt_src_tgt,worker_count => 4);
__OUTPUT__
remotecopyschema_nb
-----
                2
(1 row)
```

The completed status of the job is shown by the following:

```
tgtdb=# SELECT edb_util.process_status_from_log(clone_rmt_src_tgt);
__OUTPUT__
                                process_status_from_log
-----
↪-----
(FINISH,"29-JUN-17 current:16:00.100284 -04:00",3849,INFO,"STAGE: FINAL","successfully_
↪clone schema into tgtschema")
(1 row)
```

The following removes the log file and the pgAgent job:

```

tgtb=# SELECT edb_util.remove_log_file_and_job (clone_rmt_src_tgt,2);
__OUTPUT__
remove_log_file_and_job
-----
t
(1 row)

```

8.2.13 process_status_from_log

The `process_status_from_log` function provides the status of a cloning function from its log file.

```

process_status_from_log (
    <log_file> TEXT
)

```

The function returns the following fields from the log file:

Field name	Description
<i>status</i>	Displays either <i>STARTING</i> , <i>RUNNING</i> , <i>FINISH</i> , or <i>FAILED</i> .
<i>execution_time</i>	When the command was executed. Displayed in timestamp format.
<i>pid</i>	Session process ID in which clone schema is getting called.
<i>level</i>	Displays either <i>INFO</i> , <i>ERROR</i> , or <i>SUCCESSFUL</i> .
<i>stage</i>	Displays either <i>STARTUP</i> , <i>INITIAL</i> , <i>DDL-COLLECTION</i> , <i>PRE-DATA</i> , <i>DATA-COPY</i> , <i>POST-DATA</i> , or <i>FINAL</i> .
<i>message</i>	Information respective to each command or failure.

8.2.14 Parameters

`log_file`

Name of the log file recording the cloning of a schema as specified when the cloning function was invoked.

8.2.15 Example

The following shows usage of the `process_status_from_log` function:

```

edb=# SELECT edb_util.process_status_from_log(clone_edb_edbcopy);
__OUTPUT__
                                     process_status_from_log
-----
↪ -----
↪ (FINISH,"26-JUN-17 11:57:03.214458 -04:00",3691,INFO,"STAGE: FINAL","successfully_
↪ cloned schema")
(1 row)

```

8.2.16 remove_log_file_and_job

The `remove_log_file_and_job` function performs cleanup tasks by removing the log files created by the schema cloning functions and the jobs created by the non-blocking functions.

```
remove_log_file_and_job (
  { <log_file> TEXT |
    <job_id> INTEGER |
    <log_file> TEXT, <job_id> INTEGER
  }
)
```

Values for any or both of the two parameters may be specified when invoking the `remove_log_file_and_job` function:

- If only `log_file` is specified, then the function removes only the log file.
- If only `job_id` is specified, then the function removes only the job.
- If both are specified, then the function removes only the log file and the job.

8.2.17 Parameters

`log_file`

Name of the log file to be removed.

`job_id`

Job ID of the job to be removed.

8.2.18 Example

The following examples removes only the log file, given the log filename.

```
edb=# SELECT edb_util.remove_log_file_and_job (clone_edb_edbcopy);
__OUTPUT__
remove_log_file_and_job
-----
t
(1 row)
```

The following example removes only the job, given the job ID.

```
edb=# SELECT edb_util.remove_log_file_and_job (3);
__OUTPUT__
remove_log_file_and_job
-----
t
(1 row)
```

The following example removes the log file and the job, given both values:

```
tgtdb=# SELECT edb_util.remove_log_file_and_job (clone_rmt_src_tgt,2);
__OUTPUT__
remove_log_file_and_job
- -----
t
(1 row)
```

ENHANCED SQL AND OTHER MISCELLANEOUS FEATURES

EDB Postgres Advanced Server includes enhanced SQL functionality and various other features that provide additional flexibility and convenience. This chapter discusses some of these additions.

9.1 COMMENT

In addition to commenting on objects supported by the PostgreSQL COMMENT command, EDB Postgres Advanced Server supports comments on additional object types. The complete supported syntax is:

```
COMMENT ON
{
  AGGREGATE <aggregate_name> ( <aggregate_signature> ) |
  CAST (<source_type> AS <target_type>) |
  COLLATION <object_name> |
  COLUMN <relation_name>.<column_name> |
  CONSTRAINT <constraint_name> ON <table_name> |
  CONSTRAINT <constraint_name> ON DOMAIN <domain_name> |
  CONVERSION <object_name> |
  DATABASE <object_name> |
  DOMAIN <object_name> |
  EXTENSION <object_name> |
  EVENT TRIGGER <object_name> |
  FOREIGN DATA WRAPPER <object_name> |
  FOREIGN TABLE <object_name> |
  FUNCTION <func_name> ([[<argmode>] [<argname>] <argtype> [,...]]) |
  INDEX <object_name> |
  LARGE OBJECT <large_object_oid> |
  MATERIALIZED VIEW <object_name> |
  OPERATOR <operator_name> (left_type, right_type) |
  OPERATOR CLASS <object_name> USING <index_method> |
  OPERATOR FAMILY <object_name> USING <index_method> |
  PACKAGE <object_name>
  POLICY <policy_name> ON <table_name> |
  [ PROCEDURAL ] LANGUAGE <object_name> |
  PROCEDURE <proc_name> ([[<argmode>] [<argname>] <argtype> [, ...]]) |
  PUBLIC SYNONYM <object_name>
  ROLE <object_name> |
  RULE <rule_name> ON <table_name> |
  SCHEMA <object_name> |
  SEQUENCE <object_name> |
```

(continues on next page)

(continued from previous page)

```

SERVER <object_name> |
TABLE <object_name> |
TABLESPACE <object_name> |
TEXT SEARCH CONFIGURATION <object_name> |
TEXT SEARCH DICTIONARY <object_name> |
TEXT SEARCH PARSER <object_name> |
TEXT SEARCH TEMPLATE <object_name> |
TRANSFORM FOR <type_name> LANGUAGE <lang_name> |
TRIGGER <trigger_name> ON <table_name> |
TYPE <object_name> |
VIEW <object_name>
} IS <text>

```

where `aggregate_signature` is:

```

* |
[ <argmode> ] [ <argname> ] <argtype> [ , ... ] |
[ [ <argmode> ] [ <argname> ] <argtype> [ , ... ] ]
ORDER BY [ <argmode> ] [ <argname> ] <argtype> [ , ... ]

```

9.1.1 Parameters

`object_name`

The name of the object on which you are commenting.

AGGREGATE `aggregate_name` (`aggregate_signature`)

Include the **AGGREGATE** clause to create a comment about an aggregate. `aggregate_name` specifies the name of an aggregate, and `aggregate_signature` specifies the associated signature in one of the following forms:

```

* |
[ <argmode> ] [ <argname> ] <argtype> [ , ... ] |
[ [ <argmode> ] [ <argname> ] <argtype> [ , ... ] ]
ORDER BY [ <argmode> ] [ <argname> ] <argtype> [ , ... ]

```

Where `argmode` is the mode of a function, procedure, or aggregate argument, `argmode` may be `IN` , `OUT` , `INOUT` , or `VARIADIC` . If omitted, the default is `IN` .

`argname` is the name of an aggregate argument.

`argtype` is the data type of an aggregate argument.

CAST (`source_type` **AS** `target_type`)

Include the **CAST** clause to create a comment about a cast. When creating a comment about a cast, `source_type` specifies the source data type of the cast, and `target_type` specifies the target data type of the cast.

COLUMN `relation_name.column_name`

Include the **COLUMN** clause to create a comment about a column. `column_name` specifies name of the column to which the comment applies. `relation_name` is the table, view, composite type, or foreign table in which a column resides.

CONSTRAINT `constraint_name` **ON** `table_name`

CONSTRAINT `constraint_name` **ON** **DOMAIN** `domain_name`

Include the `CONSTRAINT` clause to add a comment about a constraint. When creating a comment about a constraint, `constraint_name` specifies the name of the constraint. `table_name` or `domain_name` specifies the name of the table or domain on which the constraint is defined.

```
FUNCTION func_name ([[argmode] [argname] argtype [, ...]])
```

Include the `FUNCTION` clause to add a comment about a function. `func_name` specifies the name of the function. `argmode` specifies the mode of the function. `argmode` may be `IN`, `OUT`, `INOUT`, or `VARIADIC`. If omitted, the default is `IN`.

`argname` specifies the name of a function, procedure, or aggregate argument. `argtype` specifies the data type of a function, procedure, or aggregate argument.

`large_object_oid`

`large_object_oid` is the system-assigned OID of the large object about which you are commenting.

```
OPERATOR operator_name (left_type, right_type)
```

Include the `OPERATOR` clause to add a comment about an operator. `operator_name` specifies the (optionally schema-qualified) name of an operator on which you are commenting. `left_type` and `right_type` are the (optionally schema-qualified) data type(s) of the operator's arguments.

```
OPERATOR CLASS object_name USING index_method
```

Include the `OPERATOR CLASS` clause to add a comment about an operator class. `object_name` specifies the (optionally schema-qualified) name of an operator on which you are commenting. `index_method` specifies the associated index method of the operator class.

```
OPERATOR FAMILY object_name USING index_method
```

Include the `OPERATOR FAMILY` clause to add a comment about an operator family. `object_name` specifies the (optionally schema-qualified) name of an operator family on which you are commenting. `index_method` specifies the associated index method of the operator family.

```
POLICY policy_name ON table_name
```

Include the `POLICY` clause to add a comment about a policy. `policy_name` specifies the name of the policy, and `table_name` specifies the table that the policy is associated with.

```
PROCEDURE proc_name ([[argmode] [argname] argtype [, ...]])
```

Include the `PROCEDURE` clause to add a comment about a procedure. `proc_name` specifies the name of the procedure. `argmode` specifies the mode of the procedure. `argmode` may be `IN`, `OUT`, `INOUT`, or `VARIADIC`. If omitted, the default is `IN`.

`argname` specifies the name of a function, procedure, or aggregate argument. `argtype` specifies the data type of a function, procedure, or aggregate argument.

```
RULE rule_name ON table_name
```

Include the `RULE` clause to specify a `COMMENT` on a rule. `rule_name` specifies the name of the rule, and `table_name` specifies the name of the table on which the rule is defined.

```
TRANSFORM FOR type_name LANGUAGE lang_name
```

Include the `TRANSFORM FOR` clause to specify a `COMMENT` on a `TRANSFORM`.

`type_name` specifies the name of the data type of the transform and `lang_name` specifies the name of the language of the transform.

```
TRIGGER trigger_name ON table_name
```

Include the `TRIGGER` clause to specify a `COMMENT` on a trigger. `trigger_name` specifies the name of the trigger, and `table_name` specifies the name of the table on which the trigger is defined.

text

The comment, written as a string literal or NULL to drop the comment.

9.1.2 Notes

Names of tables, aggregates, collations, conversions, domains, foreign tables, functions, indexes, operators, operator classes, operator families, packages, procedures, sequences, text search objects, types, and views can be schema-qualified.

9.1.3 Example

The following example adds a comment to a table named `new_emp` :

```
COMMENT ON TABLE new_emp IS This table contains information about new
employees.;
```

For more information about using the `COMMENT` command, see the PostgreSQL core documentation at:

<https://www.postgresql.org/docs/current/static/sql-comment.html>

9.2 Output of function `version()`

The text string output of the `version()` function displays the name of the product, its version, and the host system on which it has been installed.

For EDB Postgres Advanced Server, the `version()` output is in a format similar to the PostgreSQL community version in that the first text word is *PostgreSQL* instead of *EnterpriseDB* as in EDB Postgres Advanced Server version 10 and earlier.

The general format of the `version()` output is the following:

```
PostgreSQL $PG_VERSION_EXT (EnterpriseDB EDB Postgres Advanced Server $PG_VERSION) on
↪ $host
```

So for the current EDB Postgres Advanced Server the version string appears as follows:

```
edb@45032=#select version();
__OUTPUT__
version
-----
↪ -----
-----
PostgreSQL 14.0 (EnterpriseDB EDB Postgres Advanced Server 14.0.0) on x86_64-pc-linux-
↪ gnu, compiled by gcc
(GCC) 4.8.5 20150623 (Red Hat 4.8.5-11), 64-bit
(1 row)
```

In contrast, for EDB Postgres Advanced Server 10, the version string was the following:

```
edb=# select version();
__OUTPUT__
```

(continues on next page)

(continued from previous page)

version

↪ --

EnterpriseDB 10.4.9 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.4.7 20120313 (Red Hat 4.4.7-18), 64-bit (1 row)

9.3 Logical decoding on standby

Logical decoding on a standby server allows you to create a logical replication slot on a standby server that can respond to API operations such as `get`, `peek`, `advance`, etc..

For more information about the LOGICAL DECODING , refer to the
[PostgreSQL core documentation](#) .

For a logical slot on a standby server to work, the `hot_standby_feedback` parameter must be set to `ON` on the standby. The `hot_standby_feedback` parameter prevents `VACUUM` from removing recently-dead rows that are required by an existing logical replication slot on the standby server. If a slot conflict occurs on the standby, the slots are dropped.

For logical decoding on a standby to work, `wal_level` must be set to `logical` on both the primary and standby server. If `wal_level` is set to a value other than `logical` , then slots are not created. If you set `wal_level` to a value other than `logical` on primary and if there are existing logical slots on standby, such slots are dropped and new slots cannot be created.

When transactions are written to the primary server, the activity triggers the creation of a logical slot on the standby server. If a primary server is idle, creating a logical slot on a standby server may take noticeable time.

For more information about functions that support replication and logical decoding example, see to the PostgreSQL documentation available at:

<https://www.postgresql.org/docs/current/functions-admin.html#FUNCTIONS-REPLICATION>

<https://www.postgresql.org/docs/current/logicaldecoding-example.html>

SYSTEM CATALOG TABLES

The following system catalog tables contain definitions of database objects. The layout of the system tables is subject to change, if you are writing an application that depends on information stored in the system tables, it would be prudent to use an existing catalog view, or create a catalog view to isolate the application from changes to the system table.

10.1 edb_dir

The `edb_dir` table contains one row for each alias that points to a directory created with the `CREATE DIRECTORY` command. A directory is an alias for a pathname that allows a user limited access to the host file system.

You can use a directory to fence a user into a specific directory tree within the file system. For example, the `UTL_FILE` package offers functions that permit a user to read and write files and directories in the host file system, but only allows access to paths that the database administrator has granted access to via a `CREATE DIRECTORY` command.

Column	Type	Modifiers	Description
<i>dirname</i>	<i>"name"</i>	<i>not null</i>	The name of the alias.
<i>diowner</i>	<i>oid</i>	<i>not null</i>	The OID of the user that owns the alias.
<i>dirpath</i>	<i>text</i>		The directory name to which access is granted.
<i>diracl</i>	<i>aclitem[]</i>		The access control list that determines which users may access the alias.

10.2 edb_all_resource_groups

The `edb_all_resource_groups` table contains one row for each resource group created with the `CREATE RESOURCE GROUP` command and displays the number of active processes in each resource group.

Column	Type	Modifiers	Description
<i>group_name</i>	<i>"name"</i>		The name of the resource group.
<i>active_processes</i>	<i>integer</i>		Number of currently active processes in the resource group.
<i>cpu_rate_limit</i>	<i>float8</i>		Maximum CPU rate limit for the resource group. <i>0</i> means no limit.

continues on next page

Table 2 – continued from previous page

Column	Type	Modifiers	Description
<i>per_process_cpu_rate_limit</i>	<i>float8</i>		Maximum CPU rate limit per currently active process in the resource group.
<i>dirty_rate_limit</i>	<i>float8</i>		Maximum dirty rate limit for a resource group. 0 means no limit.
<i>per_process_dirty_rate_limit</i>	<i>float8</i>		Maximum dirty rate limit per currently active process in the resource group.

10.3 edb_policy

The `edb_policy` table contains one row for each policy.

Column	Type	Modifiers	Description
<i>polycyname</i>	<i>name</i>	<i>not null</i>	The policy name.
<i>policygroup</i>	<i>oid</i>	<i>not null</i>	Currently unused.
<i>policyobject</i>	<i>oid</i>	<i>not null</i>	The OID of the table secured by this policy (the <i>object_schema</i> plus the <i>object_name</i>).
<i>policykind</i>	<i>char</i>	<i>not null</i>	The kind of object secured by this policy: 'r' for a table 'v' for a view = for a synonym Currently always 'r'.
<i>policyproc</i>	<i>oid</i>	<i>not null</i>	The OID of the policy function (<i>function_schema</i> plus <i>policy_function</i>)
<i>policyinsert</i>	<i>boolean</i>	<i>not null</i>	True if the policy is enforced by <i>INSERT</i> statements.
<i>policyselect</i>	<i>boolean</i>	<i>not null</i>	True if the policy is enforced by <i>SELECT</i> statements.
<i>policydelete</i>	<i>boolean</i>	<i>not null</i>	True if the policy is enforced by <i>DELETE</i> statements.
<i>policyupdate</i>	<i>boolean</i>	<i>not null</i>	True if the policy is enforced by <i>UPDATE</i> statements.
<i>policyindex</i>	<i>boolean</i>	<i>not null</i>	Currently unused.
<i>policyenabled</i>	<i>boolean</i>	<i>not null</i>	True if the policy is enabled.
<i>policyupdatecheck</i>	<i>boolean</i>	<i>not null</i>	True if rows updated by an <i>UPDATE</i> statement must satisfy the policy.
<i>polycystatic</i>	<i>boolean</i>	<i>not null</i>	Currently unused.
<i>policytype</i>	<i>integer</i>	<i>not null</i>	Currently unused.
<i>policyopts</i>	<i>integer</i>	<i>not null</i>	Currently unused.
<i>policyseccols</i>	<i>int2vector</i>	<i>not null</i>	The column numbers for columns listed in <i>sec_relevant_cols</i> .

10.4 edb_profile

The `edb_profile` table stores information about the available profiles. `edb_profiles` is shared across all databases within a cluster.

Column	Type	References	Description
<i>oid</i>	<i>oid</i>		Row identifier (hidden attribute, must be explicitly selected).
<i>prfname</i>	<i>name</i>		The name of the profile.
<i>prffailedloginattempts</i>	<i>integer</i>		The number of failed login attempts allowed by the profile. -1 indicates that the value from the default profile should be used. -2 indicates no limit on failed login attempts.
<i>prfpasswordlocktime</i>	<i>integer</i>		The password lock time associated with the profile (in seconds). -1 indicates that the value from the default profile should be used. -2 indicates that the account should be locked permanently.
<i>prfpasswordlifetime</i>	<i>integer</i>		The password life time associated with the profile (in seconds). -1 indicates that the value from the default profile should be used. -2 indicates that the password never expires.
<i>prfpasswordgracetime</i>	<i>integer</i>		The password grace time associated with the profile (in seconds). -1 indicates that the value from the default profile should be used. -2 indicates that the password never expires.
<i>prfpasswordreusetime</i>	<i>integer</i>		The number of seconds that a user must wait before reusing a password. -1 indicates that the value from the default profile should be used. -2 indicates that the old passwords can never be reused.
<i>prfpasswordreusemax</i>	<i>integer</i>		The number of password changes that have to occur before a password can be reused. -1 indicates that the value from the default profile should be used. -2 indicates that the old passwords can never be reused.
<i>prfpasswordverifyfuncdb</i>	<i>oid</i>	<i>pg_database.oid</i>	The OID of the database in which the password verify function exists.
<i>prfpasswordverifyfunc</i>	<i>oid</i>	<i>pg_proc.oid</i>	The OID of the password verify function associated with the profile.

10.5 edb_redaction_column

The catalog `edb_redaction_column` stores information of data redaction policy attached to the columns of the table.

Column	Type	References	Description
<i>oid</i>	<i>oid</i>		Row identifier (hidden attribute, must be explicitly selected).
<i>rdpolicyid</i>	<i>oid</i>	<i>edb_redaction_policy.oid</i>	The data redaction policy applies to the described column.
<i>rdrelid</i>	<i>oid</i>	<i>pg_class.oid</i>	The table that the described column belongs to.
<i>rdattnum</i>	<i>int2</i>	<i>pg_attribute.attnum</i>	The number of the described column.
<i>rdscope</i>	<i>int2</i>		The redaction scope: 1 = query, 2 = top_tlist, 4 = top_tlist_or_error
<i>rdexception</i>	<i>int2</i>		The redaction exception: 8 = none, 16 = equal, 32 = leakproof
<i>rdfuncexpr</i>	<i>pg_node_tree</i>		Data redaction function expression

Note: The described column is redacted if the redaction policy *edb_redaction_column.rdpolicyid* on the table is enabled and the redaction policy expression *edb_redaction_policy.rdexpr* evaluates to true.

10.6 edb_redaction_policy

The catalog `edb_redaction_policy` stores information of the redaction policies for tables.

Column	Type	References	Description
<i>oid</i>	<i>oid</i>		Row identifier (hidden attribute, must be explicitly selected).
<i>rdname</i>	<i>name</i>		The name of the data redaction policy
<i>rdrelid</i>	<i>oid</i>	<i>pg_class.oid</i>	The table to which the data redaction policy applies.
<i>rdenable</i>	<i>boolean</i>		Is the data redaction policy enabled?
<i>rdexpr</i>	<i>pg_node_tree</i>		The data redaction policy expression.

Note: The data redaction policy applies for the table if it is enabled and the expression ever evaluated true.

10.7 edb_resource_group

The `edb_resource_group` table contains one row for each resource group created with the `CREATE RESOURCE GROUP` command.

Column	Type	Modifiers	Description
<i>rgrpname</i>	<i>"name"</i>	<i>not null</i>	The name of the resource group.
<i>rgrpcpuratelimt</i>	<i>float8</i>	<i>not null</i>	Maximum CPU rate limit for a resource group. 0 means no limit.
<i>rgrpdirtyratelimt</i>	<i>float8</i>	<i>not null</i>	Maximum dirty rate limit for a resource group. 0 means no limit.

10.8 edb_variable

The `edb_variable` table contains one row for each package level variable (each variable declared within a package).

Column	Type	Modifiers	Description
<i>varname</i>	<i>"name"</i>	<i>not null</i>	The name of the variable.
<i>varpackage</i>	<i>oid</i>	<i>not null</i>	The OID of the <i>pg_namespace</i> row that stores the package.
<i>vartype</i>	<i>oid</i>	<i>not null</i>	The OID of the <i>pg_type</i> row that defines the type of the variable.
<i>varaccess</i>	<i>"char"</i>	<i>not null</i>	+ if the variable is visible outside of the package. - if the variable is only visible within the package. Note: Public variables are declared within the package header, private variables are declared within the package body.
<i>varsrc</i>	<i>text</i>		Contains the source of the variable declaration, including any default value expressions for the variable.
<i>vartseq</i>	<i>smallint</i>	<i>not null</i>	The order in which the variable was declared in the package.

10.9 pg_synonym

The `pg_synonym` table contains one row for each synonym created with the `CREATE SYNONYM` command or `CREATE PUBLIC SYNONYM` command.

Column	Type	Modifiers	Description
<i>synname</i>	<i>"name"</i>	<i>not null</i>	The name of the synonym.
<i>synnamespace</i>	<i>oid</i>	<i>not null</i>	Replaces <i>synowner</i> . Contains the OID of the <i>pg_namespace</i> row where the synonym is stored
<i>synowner</i>	<i>oid</i>	<i>not null</i>	The OID of the user that owns the synonym.

continues on next page

Table 9 – continued from previous page

Column	Type	Modifiers	Description
<i>synobjschema</i>	<i>“name”</i>	<i>not null</i>	The schema in which the referenced object is defined.
<i>synobjname</i>	<i>“name”</i>	<i>not null</i>	The name of the referenced object.
<i>synlink</i>	<i>text</i>		The (optional) name of the database link in which the referenced object is defined.

10.10 product_component_version

The `product_component_version` table contains information about feature compatibility, an application can query this table at installation or run time to verify that features used by the application are available with this deployment.

Column	Type	Description
<i>product</i>	<i>character varying (74)</i>	The name of the product.
<i>version</i>	<i>character varying (74)</i>	The version number of the product.
<i>status</i>	<i>character varying (74)</i>	The status of the release.

EDB POSTGRES ADVANCED SERVER EXCEPTIONS

The following table lists the predefined exceptions, the SQLstate values, associated redwood error code, and description of the exceptions.

Exception name	SQLState	Redwood error code	Description
<i>value_error</i>	22000	-6502	The exception occurs when the conversion of a character string to a number fails.
<i>invalid_number</i>	22000	-6502	The exception is raised in PL statement, when the conversion of a character string to number fails.
<i>date-time_value_out_of_range</i>	22008	-1863	The exception occurs while writing a field in date format, which is outside the valid range.
<i>divide_by_zero</i>	22012	-1476	The exception occurs when a program attempts to divide a number by zero.
<i>zero_divide</i>	22012	-1476	The exception occurs when a program attempts to divide a number by zero.
<i>dup_val_on_index</i>	23505	-1	The exception occurs when a program attempts to store duplicate values in a column that is constrained by a unique index.
<i>invalid_cursor</i>	34000	-1001	The exception occurs when a program attempts a cursor operation on an invalid cursor, such as closing an unopened cursor.
<i>cursor_already_open</i>	42P03	-6511	The exception occurs when a program attempts to open an already open cursor.
<i>collection_is_null</i>	P1403	-6531	The exception occurs when a program attempts to assign values to the elements of nested table or varray that are uninitialized.

continues on next page

Table 1 – continued from previous page

Exception name	SQLState	Redwood error code	Description
<i>subscript_beyond_count</i>	P1404	-6533	The exception occurs when a program attempts to reference a nested table or varray using an index number larger than the number of elements in the collection.
<i>subscript_outside_limit</i>	P1405	-6532	The exception occurs when a program attempts to reference a nested table or varray element using an index number that is outside the range.

DBMS_CRYPTO package	
<i>ciphersuiteinvalid</i>	The exception occurs when the cipher suite is not defined.
<i>ciphersuitenull</i>	The exception occurs when no value is specified for the cipher suite or contains a <i>NULL</i> value.
<i>keybadsize</i>	The exception occurs when the specified key size is bad.
<i>keynull</i>	The exception occurs when the key is not specified.
UTL_FILE package	
<i>invalid_filehandle</i>	The exception occurs when file handle is invalid.
<i>invalid_maxlinesize</i>	The exception occurs when the max line size is invalid or the max line size value is not within the range.
<i>invalid_mode</i>	The exception occurs when the <i>open_mode</i> parameter in <i>FOPEN</i> is invalid.
<i>invalid_operation</i>	The exception occurs when the file could not be opened or used upon request.
<i>invalid_path</i>	The exception occurs when the file location or filename is invalid.
<i>read_error</i>	The exception occurs when the operating system error occurred during the read operation.
<i>write_error</i>	The exception occurs when the operating system error occurred during the write operation.
UTL_HTTP package	
<i>end_of_body</i>	The exception occurs when the end of HTTP response body is reached.
UTL_URL package	
<i>bad_fixed_width_charset</i>	The exception occurs when the fixed-width multibyte character is not allowed as a URL character set.
<i>bad_url</i>	The exception occurs when the URL includes badly formed escape code sequences.

11.1 EDB Postgres Advanced Server keywords

A keyword is a word that is recognized by the EDB Postgres Advanced Server parser as having a special meaning or association. You can use the `pg_get_keywords()` function to retrieve an up-to-date list of the EDB Postgres Advanced Server keywords:

```
acctg=#
acctg=# SELECT * FROM pg_get_keywords();
__OUTPUT__
      word      | catcode |      catdesc
-----+-----+-----
  abort         | U       | unreserved
```

(continues on next page)

(continued from previous page)

absolute	U	unreserved
access	U	unreserved
...		

`pg_get_keywords` returns a table containing the keywords recognized by EDB Postgres Advanced Server:

- The `word` column displays the keyword.
- The `catcode` column displays a category code.
- The `catdesc` column displays a brief description of the category to which the keyword belongs.

Note that any character can be used in an identifier if the name is enclosed in double quotes. You can selectively query the `pg_get_keywords()` function to retrieve an up-to-date list of the EDB Postgres Advanced Server keywords that belong to a specific category:

```
SELECT * FROM pg_get_keywords() WHERE catcode = code;
```

Where `code` is:

R - The word is reserved. Reserved keywords may never be used as an identifier, they are reserved for use by the server.

U - The word is unreserved. Unreserved words are used internally in some contexts, but may be used as a name for a database object.

T - The word is used internally, but may be used as a name for a function or type.

C - The word is used internally, and may not be used as a name for a function or type.

For more information about EDB Postgres Advanced Server identifiers and keywords, see to the PostgreSQL core documentation at:

<https://www.postgresql.org/docs/current/static/sql-syntax-lexical.html>