Postgres Plus® Advanced Server Guide

Postgres Plus Advanced Server 9.2

February 1, 2013

Postgres Plus Advanced Server Guide, Version 4.0
by EnterpriseDB Corporation

EnterpriseDB Corporation, 34 Crosby Drive, Suite 100, Bedford, MA 01730, USA
T +1 781 357 3390 **F** +1 978 589 5701 **E** info@enterprisedb.com www.enterprisedb.com

Table of Contents

1 Introduction

This guide describes features of *Postgres Plus Advanced Server 9.2*. Some of these features apply to PostgreSQL® as well, but many are features unique to Advanced Server.

Within this guide you will find information about the following:

- Setting the database server configuration parameters to control various behavioral and environmental characteristics of Advanced Server. See Section 2.2 for more information.
- Utilizing the auditing capabilities of Advanced Server with the audit logs and the `edb_audit` configuration parameters. See Section 2.3 for more information.
- Using SQL/Protect to guard against SQL injection attacks. See Chapter 3 for more information.
- Using libpq to connect C applications with the server. See Section 4.1 for more information.
- Using the Debugger to aid in the development of SPL and PL/pgSQL applications. See Section 4.2 for information.

This guide 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.

Throughout this guide, the directory path of Postgres Plus Advanced Server is referred to as `POSTGRES_PLUS_HOME`.

> For Linux installations, the default directory path is `/opt/PostgresPlus/version_no`.
>
> For Windows installations, the default directory path is `C:\Program Files\PostgresPlus\version_no`

Where *`version_no`* is the Advanced Server version number.

1.1 Typographical Conventions Used in this Guide

Certain typographical conventions are used in this manual to clarify the meaning and usage of various commands, statements, programs, examples, etc. This section provides a summary of these conventions.

In the following descriptions a *term* refers to any word or group of words which may be language keywords, user-supplied values, literals, etc. A term's exact meaning depends upon the context in which it is used.

- *Italic font* introduces a new term, typically, in the sentence that defines it for the first time.
- `Fixed-width (mono-spaced) font` is used for terms that must be given literally such as SQL commands, specific table and column names used in the examples, programming language keywords, directory paths and file names, parameter values, etc. For example `postgresql.conf`, `SELECT * FROM emp;`
- *`Italic fixed-width font`* is used for terms for which the user must substitute values in actual usage. For example, `DELETE FROM` *`table_name`*`;`
- A vertical pipe | denotes a choice between the terms on either side of the pipe. A vertical pipe is used to separate two or more alternative terms within square brackets (optional choices) or braces (one mandatory choice).
- Square brackets [ ] denote that one or none of the enclosed term(s) may be substituted. For example, `[ a | b ]`, means choose one of "`a`" or "`b`" or neither of the two.
- Braces {} denote that exactly one of the enclosed alternatives must be specified. For example, `{ a | b }`, means exactly one of "`a`" or "`b`" must be specified.
- Ellipses ... denote that the proceeding term may be repeated. For example, `[ a | b ] ...` means that you may have the sequence, "`b a a b a`".

1.2 About the Examples Used in this Guide

The examples in this guide are shown in the type and background illustrated below.

```
Examples and output from examples are shown in fixed-width, blue font on a
light blue background.
```

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

The tables and programs in the sample database can be re-created at any time by executing the script, `edb-sample.sql`, located in directory *`POSTGRES_PLUS_HOME`*`/installer/server`.

This 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 will be 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 by issuing the command:

```
SHOW SEARCH_PATH;
```

Altering the search path can be done using commands in PSQL.

1.2.1.1 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, etc. 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 `edb-sample.sql` script:

```
--
--  Script that creates the 'sample' tables, views, procedures,
--  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          NUMBER(2) NOT NULL CONSTRAINT dept_pk PRIMARY KEY,
    dname           VARCHAR2(14) CONSTRAINT dept_dname_uq UNIQUE,
    loc             VARCHAR2(13)
);
--
--  Create the 'emp' table
--
CREATE TABLE emp (
    empno           NUMBER(4) NOT NULL CONSTRAINT emp_pk PRIMARY KEY,
    ename           VARCHAR2(10),
    job             VARCHAR2(9),
    mgr             NUMBER(4),
    hiredate        DATE,
    sal             NUMBER(7,2) CONSTRAINT emp_sal_ck CHECK (sal > 0),
    comm            NUMBER(7,2),
    deptno          NUMBER(2) CONSTRAINT emp_ref_dept_fk
                        REFERENCES dept(deptno)
);
--
--  Create the 'jobhist' table
--
CREATE TABLE jobhist (
    empno           NUMBER(4) NOT NULL,
    startdate       DATE NOT NULL,
    enddate         DATE,
    job             VARCHAR2(9),
    sal             NUMBER(7,2),
    comm            NUMBER(7,2),
    deptno          NUMBER(2),
    chgdesc         VARCHAR2(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);
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;
--
--  Procedure that lists all employees' numbers and names
--  from the 'emp' table using a cursor.
--
CREATE OR REPLACE PROCEDURE list_emp
IS
    v_empno         NUMBER(4);
    v_ename         VARCHAR2(10);
    CURSOR emp_cur IS
        SELECT empno, ename FROM emp ORDER BY empno;
BEGIN
    OPEN emp_cur;
    DBMS_OUTPUT.PUT_LINE('EMPNO    ENAME');
    DBMS_OUTPUT.PUT_LINE('-----    -------');
    LOOP
        FETCH emp_cur INTO v_empno, v_ename;
        EXIT WHEN emp_cur%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(v_empno || '     ' || v_ename);
    END LOOP;
    CLOSE emp_cur;
END;
/
--
--  Procedure that selects an employee row given the employee
--  number and displays certain columns.
--
CREATE OR REPLACE PROCEDURE select_emp (
    p_empno         IN  NUMBER
)
IS
    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     VARCHAR2(10);
BEGIN
```

```
    SELECT ename, hiredate, sal, NVL(comm, 0), dname
        INTO v_ename, v_hiredate, v_sal, v_comm, v_dname
        FROM emp e, dept d
        WHERE empno = p_empno
          AND e.deptno = d.deptno;
    v_disp_date := TO_CHAR(v_hiredate, 'MM/DD/YYYY');
    DBMS_OUTPUT.PUT_LINE('Number    : ' || p_empno);
    DBMS_OUTPUT.PUT_LINE('Name      : ' || v_ename);
    DBMS_OUTPUT.PUT_LINE('Hire Date : ' || v_disp_date);
    DBMS_OUTPUT.PUT_LINE('Salary    : ' || v_sal);
    DBMS_OUTPUT.PUT_LINE('Commission: ' || v_comm);
    DBMS_OUTPUT.PUT_LINE('Department: ' || v_dname);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Employee ' || p_empno || ' not found');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('The following is SQLERRM:');
        DBMS_OUTPUT.PUT_LINE(SQLERRM);
        DBMS_OUTPUT.PUT_LINE('The following is SQLCODE:');
        DBMS_OUTPUT.PUT_LINE(SQLCODE);
END;
/
--
--  Procedure that queries the 'emp' table based on
--  department number and employee number or name.  Returns
--  employee number and name as IN OUT parameters and job,
--  hire date, and salary as OUT parameters.
--
CREATE OR REPLACE PROCEDURE emp_query (
    p_deptno        IN     NUMBER,
    p_empno         IN OUT NUMBER,
    p_ename         IN OUT VARCHAR2,
    p_job           OUT    VARCHAR2,
    p_hiredate      OUT    DATE,
    p_sal           OUT    NUMBER
)
IS
BEGIN
    SELECT empno, ename, job, hiredate, sal
        INTO p_empno, p_ename, p_job, p_hiredate, p_sal
        FROM emp
        WHERE deptno = p_deptno
          AND (empno = p_empno
           OR  ename = UPPER(p_ename));
END;
/
--
--  Procedure to call 'emp_query_caller' with IN and IN OUT
--  parameters.  Displays the results received from IN OUT and
--  OUT parameters.
--
CREATE OR REPLACE PROCEDURE emp_query_caller
IS
    v_deptno        NUMBER(2);
    v_empno         NUMBER(4);
    v_ename         VARCHAR2(10);
    v_job           VARCHAR2(9);
    v_hiredate      DATE;
    v_sal           NUMBER;
BEGIN
    v_deptno := 30;
    v_empno  := 0;
    v_ename  := 'Martin';
```

```
    emp_query(v_deptno, v_empno, v_ename, v_job, v_hiredate, v_sal);
    DBMS_OUTPUT.PUT_LINE('Department : ' || v_deptno);
    DBMS_OUTPUT.PUT_LINE('Employee No: ' || v_empno);
    DBMS_OUTPUT.PUT_LINE('Name       : ' || v_ename);
    DBMS_OUTPUT.PUT_LINE('Job        : ' || v_job);
    DBMS_OUTPUT.PUT_LINE('Hire Date  : ' || v_hiredate);
    DBMS_OUTPUT.PUT_LINE('Salary     : ' || v_sal);
EXCEPTION
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('More than one employee was selected');
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No employees were selected');
END;
/
--
--  Function to compute yearly compensation based on semimonthly
--  salary.
--
CREATE OR REPLACE FUNCTION emp_comp (
    p_sal           NUMBER,
    p_comm          NUMBER
) RETURN NUMBER
IS
BEGIN
    RETURN (p_sal + NVL(p_comm, 0)) * 24;
END;
/
--
--  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 RETURN NUMBER
IS
    v_cnt           INTEGER := 1;
    v_new_empno     NUMBER;
BEGIN
    WHILE v_cnt > 0 LOOP
        SELECT next_empno.nextval INTO v_new_empno FROM dual;
        SELECT COUNT(*) INTO v_cnt FROM emp WHERE empno = v_new_empno;
    END LOOP;
    RETURN v_new_empno;
END;
/
--
--  EDB-SPL function that adds a new clerk to table 'emp'.  This function
--  uses package 'emp_admin'.
--
CREATE OR REPLACE FUNCTION hire_clerk (
    p_ename         VARCHAR2,
    p_deptno        NUMBER
) RETURN NUMBER
IS
    v_empno         NUMBER(4);
    v_ename         VARCHAR2(10);
    v_job           VARCHAR2(9);
    v_mgr           NUMBER(4);
    v_hiredate      DATE;
    v_sal           NUMBER(7,2);
    v_comm          NUMBER(7,2);
    v_deptno        NUMBER(2);
BEGIN
    v_empno := new_empno;
    INSERT INTO emp VALUES (v_empno, p_ename, 'CLERK', 7782,
```

```
        TRUNC(SYSDATE), 950.00, NULL, p_deptno);
    SELECT empno, ename, job, mgr, hiredate, sal, comm, deptno INTO
        v_empno, v_ename, v_job, v_mgr, v_hiredate, v_sal, v_comm, v_deptno
        FROM emp WHERE empno = v_empno;
    DBMS_OUTPUT.PUT_LINE('Department : ' || v_deptno);
    DBMS_OUTPUT.PUT_LINE('Employee No: ' || v_empno);
    DBMS_OUTPUT.PUT_LINE('Name       : ' || v_ename);
    DBMS_OUTPUT.PUT_LINE('Job        : ' || v_job);
    DBMS_OUTPUT.PUT_LINE('Manager    : ' || v_mgr);
    DBMS_OUTPUT.PUT_LINE('Hire Date  : ' || v_hiredate);
    DBMS_OUTPUT.PUT_LINE('Salary     : ' || v_sal);
    DBMS_OUTPUT.PUT_LINE('Commission : ' || v_comm);
    RETURN v_empno;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('The following is SQLERRM:');
        DBMS_OUTPUT.PUT_LINE(SQLERRM);
        DBMS_OUTPUT.PUT_LINE('The following is SQLCODE:');
        DBMS_OUTPUT.PUT_LINE(SQLCODE);
        RETURN -1;
END;
/
--
--  PostgreSQL PL/pgSQL 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,
        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:';
        RAISE INFO '%', SQLERRM;
        RAISE INFO 'The following is SQLSTATE:';
```

```
        RAISE INFO '%', 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 TRIGGER user_audit_trig
    AFTER INSERT OR UPDATE OR DELETE ON emp
DECLARE
    v_action        VARCHAR2(24);
BEGIN
    IF INSERTING THEN
        v_action := ' added employee(s) on ';
    ELSIF UPDATING THEN
        v_action := ' updated employee(s) on ';
    ELSIF DELETING THEN
        v_action := ' deleted employee(s) on ';
    END IF;
    DBMS_OUTPUT.PUT_LINE('User ' || USER || v_action ||
TO_CHAR(SYSDATE,'YYYY-MM-DD'));
END;
/
--
--  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 TRIGGER emp_sal_trig
    BEFORE DELETE OR INSERT OR UPDATE ON emp
    FOR EACH ROW
DECLARE
    sal_diff       NUMBER;
BEGIN
    IF INSERTING THEN
        DBMS_OUTPUT.PUT_LINE('Inserting employee ' || :NEW.empno);
```

```
        DBMS_OUTPUT.PUT_LINE('..New salary: ' || :NEW.sal);
    END IF;
    IF UPDATING THEN
        sal_diff := :NEW.sal - :OLD.sal;
        DBMS_OUTPUT.PUT_LINE('Updating employee ' || :OLD.empno);
        DBMS_OUTPUT.PUT_LINE('..Old salary: ' || :OLD.sal);
        DBMS_OUTPUT.PUT_LINE('..New salary: ' || :NEW.sal);
        DBMS_OUTPUT.PUT_LINE('..Raise     : ' || sal_diff);
    END IF;
    IF DELETING THEN
        DBMS_OUTPUT.PUT_LINE('Deleting employee ' || :OLD.empno);
        DBMS_OUTPUT.PUT_LINE('..Old salary: ' || :OLD.sal);
    END IF;
END;
/
--
--  Package specification for the 'emp_admin' package.
--
CREATE OR REPLACE PACKAGE emp_admin
IS
    FUNCTION get_dept_name (
        p_deptno        NUMBER
    ) RETURN VARCHAR2;
    FUNCTION update_emp_sal (
        p_empno         NUMBER,
        p_raise         NUMBER
    ) RETURN NUMBER;
    PROCEDURE hire_emp (
        p_empno         NUMBER,
        p_ename         VARCHAR2,
        p_job           VARCHAR2,
        p_sal           NUMBER,
        p_hiredate      DATE,
        p_comm          NUMBER,
        p_mgr           NUMBER,
        p_deptno        NUMBER
    );
    PROCEDURE fire_emp (
        p_empno         NUMBER
    );
END emp_admin;
/
--
--  Package body for the 'emp_admin' package.
--
CREATE OR REPLACE PACKAGE BODY emp_admin
IS
    --
    --  Function that queries the 'dept' table based on the department
    --  number and returns the corresponding department name.
    --
    FUNCTION get_dept_name (
        p_deptno        IN NUMBER
    ) RETURN VARCHAR2
    IS
        v_dname         VARCHAR2(14);
    BEGIN
        SELECT dname INTO v_dname FROM dept WHERE deptno = p_deptno;
        RETURN v_dname;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Invalid department number ' || p_deptno);
            RETURN '';
```

```
    END;
    --
    --  Function that updates an employee's salary based on the
    --  employee number and salary increment/decrement passed
    --  as IN parameters.  Upon successful completion the function
    --  returns the new updated salary.
    --
    FUNCTION update_emp_sal (
        p_empno         IN NUMBER,
        p_raise         IN NUMBER
    ) RETURN NUMBER
    IS
        v_sal           NUMBER := 0;
    BEGIN
        SELECT sal INTO v_sal FROM emp WHERE empno = p_empno;
        v_sal := v_sal + p_raise;
        UPDATE emp SET sal = v_sal WHERE empno = p_empno;
        RETURN v_sal;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Employee ' || p_empno || ' not found');
            RETURN -1;
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('The following is SQLERRM:');
            DBMS_OUTPUT.PUT_LINE(SQLERRM);
            DBMS_OUTPUT.PUT_LINE('The following is SQLCODE:');
            DBMS_OUTPUT.PUT_LINE(SQLCODE);
            RETURN -1;
    END;
    --
    --  Procedure that inserts a new employee record into the 'emp' table.
    --
    PROCEDURE hire_emp (
        p_empno         NUMBER,
        p_ename         VARCHAR2,
        p_job           VARCHAR2,
        p_sal           NUMBER,
        p_hiredate      DATE,
        p_comm          NUMBER,
        p_mgr           NUMBER,
        p_deptno        NUMBER
    )
    AS
    BEGIN
        INSERT INTO emp(empno, ename, job, sal, hiredate, comm, mgr, deptno)
            VALUES(p_empno, p_ename, p_job, p_sal,
                   p_hiredate, p_comm, p_mgr, p_deptno);
    END;
    --
    --  Procedure that deletes an employee record from the 'emp' table based
    --  on the employee number.
    --
    PROCEDURE fire_emp (
        p_empno         NUMBER
    )
    AS
    BEGIN
        DELETE FROM emp WHERE empno = p_empno;
    END;
END;
/
COMMIT;
```

2 Database Administration

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

2.1 Creating a New Cluster

A database cluster is a collection of databases, users and groups. A cluster is distinguished by the port on which the server listens for client connections, and the directory in which the data files are stored. You can use the `initdb` command to initialize a new database cluster on your Advanced Server host.

When you invoke the `initdb` command, you can use command options to specify details about the new cluster.

--no-redwood-compat

The `--no-redwood-compat` option instructs Advanced Server to initialize the new cluster without complete support of Oracle-compatibility features, and configures the server to support PostgreSQL functionality. By default, Advanced Server initializes new clusters with support for Oracle-compatible features.

If you include the `--no-redwood-compat` flag when invoking `initdb`, the command will create a database named `postgres`.

The `postgresql.conf` file will enforce PostgreSQL specific behaviors:

- The `datestyle` parameter will be set to `'iso,mdy'`.
- The `edb_redwood_date` parameter will be set to `off`.
- The `edb_redwood_strings` parameter will be set to `off`.
- The `db_dialect` parameter will be set to `'postgres'`.

Oracle-compatible packages, casts, functions and views will not be installed.

If you do not include the `--no-redwood-compat` option when invoking `initdb`, your new cluster will be initialized with support for Oracle-compatible features. The new cluster will have support for Oracle-compatible casts, and the `DUAL` table. The new cluster will also provide Oracle-compatible functions and views.

If you omit the `--no-redwood-compat` flag when invoking `initdb`, the command will create two databases; one database named `edb`, and one named `postgres`. These databases are initialized with identical content. The `edb` database is deprecated, and is included for backward compatibility.

The `postgresql.conf` file enforces Oracle-compatible behaviors:

- The `datestyle` parameter is set to `'redwood,show_time'`.
- The `edb_redwood_date` is set to `on`.
- The `edb_redwood_strings` is set to `on`.
- The `db_dialect` is set to `'redwood'`.

If you do not specify the `--no-redwood-compat` flag when invoking `initdb`, the new cluster will include the Oracle-compatible packages; for a complete list of the packages and their content, please see the *Postgres Plus Advanced Server Oracle Compatibility Developer's Guide*, available at:

http://www.enterprisedb.com/PPAS_Oracle_Compatibility_Guide

For detailed information about using the `initdb` command, and other `initdb` options, please see the *PostgreSQL Core Documentation*, available from the EnterpriseDB website at:

http://www.enterprisedb.com/docs/en/9.2/pg/app-initdb.html

2.2 Configuration Parameters

This section describes the database server configuration parameters of Advanced Server. These 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, Oracle compatibility features, locale and formatting settings, and so on.

Most of these configuration parameters apply to PostgreSQL as well. Configuration parameters that apply only to Advanced Server are noted in Section 2.2.2.

Additional information about configuration parameters can be found in the *PostgreSQL Core Documentation*, available at the EnterpriseDB website at:

http://www.enterprisedb.com/docs/en/9.2/pg/index.html

2.2.1 Setting Configuration Parameters

This section provides an overview of how configuration parameters are specified and set.

Each configuration parameter is set using a name/value pair. Parameter names are case-insensitive. The parameter name is typically separated from its value by an optional equals sign (=).

The following is an example of 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 can be written as `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. Enclose in single quotes if the value is not 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 case-insensitive.

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. Default units can be found by referencing the system view `pg_settings.unit`. A different unit can be specified 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.

The configuration parameter settings can be established in a number of different ways:

- There is a number of parameter settings that are established when the Advanced Server product is built. These are read-only parameters, and their values cannot be changed. There are also a couple of parameters that are permanently set for each database when the database is created. These parameters are read-only as well and cannot be subsequently changed 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. Some of these initial parameter settings can be overridden as discussed in the following bullet points. All configuration parameters have built-in default settings that are in effect if not explicitly overridden.
- Parameter settings can be modified 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, a new session must be started to use the new settings. And yet for other parameter types, modified settings do not take effect until the database server is stopped and restarted. See Section 18.1, "Setting Parameters" in the *PostgreSQL Core Documentation* for information on how to reload the configuration file.
- The SQL commands `ALTER DATABASE`, `ALTER ROLE`, or `ALTER ROLE IN DATABASE` can be used to modify certain parameter settings. The modified parameter settings take effect for new sessions after the command is executed. `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 methods discussed in the second and third bullet points. Parameter settings established using the `ALTER DATABASE`, `ALTER ROLE`, or `ALTER ROLE IN DATABASE` commands can only be changed by: a) re-issuing these commands with a different parameter value, or b) 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 established by the methods set forth in the prior bullet points. See Section I, "SQL Commands" of Chapter VI "Reference" in the *PostgreSQL Core Documentation* for the exact syntax of these SQL commands.
- Changes can be made for certain parameter settings for the duration of individual sessions using the `PGOPTIONS` environment variable or by using the `SET` command within the EDB-PSQL or PSQL command line terminal programs. Parameter settings made in this manner override settings established using any of the methods described by the second, third, and fourth bullet points, but only for the duration of the session.

2.2.2 Summary of Configuration Parameters

This section contains a summary table listing all 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 whereby the latter settings in this list override prior ones: a) the entire database cluster, b) specific databases in the database cluster, c) specific roles, d) specific roles when connected to specific databases, e) a specific session.
- **When Takes Effect.** When a changed parameter setting takes effect. 'Preset' – Established when the Advanced Server product is built or a particular database is created. This is a read-only parameter and cannot be changed. 'Restart' – Database server must be restarted. 'Reload' – Configuration file must be reloaded (or the database server can be restarted). '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. 'PPAS service account' – Postgres Plus Advanced Server service account (`enterprisedb` for an Oracle compatible mode installation, `postgres` for a PostgreSQL compatible mode installation). 'Superuser' – Database role with superuser privileges. 'User' – Any database role with permissions on the affected database object (the database or role to be altered with the `ALTER` command). 'n/a' – Parameter setting cannot be changed by any user.
- **Description.** Brief description of the configuration parameter.
- **PPAS Only.** 'X' – Configuration parameter is applicable to Postgres Plus Advanced Server only. No entry in this column indicates the configuration parameter applies to PostgreSQL as well.

Note: There are a number of parameters that should never be altered. These are designated as "**Note: For internal use only**" in the Description column.

Table 1 - Summary of Configuration Parameters

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

Parameter	Scope of Effect	When Takes Effect	Authorized User	Description	PPAS Only
autovacuum_vacuum_scale_factor	Cluster	Reload	PPAS service account	Number of tuple updates or deletes prior to vacuum as a fraction of reltuples.	
autovacuum_vacuum_threshold	Cluster	Reload	PPAS service account	Minimum number of tuple updates or deletes prior to vacuum.	
backslash_quote	Session	Immediate	User	Sets whether "\'" is allowed in string literals.	
bgwriter_delay	Cluster	Reload	PPAS service account	Background writer sleep time between rounds.	
bgwriter_lru_maxpages	Cluster	Reload	PPAS service account	Background writer maximum number of LRU pages to flush per round.	
bgwriter_lru_multiplier	Cluster	Reload	PPAS service account	Multiple of the average buffer usage to free per round.	
block_size	Cluster	Preset	n/a	Shows the size of a disk block.	
bonjour	Cluster	Restart	PPAS service account	Enables advertising the server via Bonjour.	
bonjour_name	Cluster	Restart	PPAS service account	Sets the Bonjour service name.	
bytea_output	Session	Immediate	User	Sets the output format for bytea.	
check_function_bodies	Session	Immediate	User	Check function bodies during CREATE FUNCTION.	
checkpoint_completion_target	Cluster	Reload	PPAS service account	Time spent flushing dirty buffers during checkpoint, as fraction of checkpoint interval.	
checkpoint_segments	Cluster	Reload	PPAS service account	Sets the maximum distance in log segments between automatic WAL checkpoints.	
checkpoint_timeout	Cluster	Reload	PPAS service account	Sets the maximum time between automatic WAL checkpoints.	
checkpoint_warning	Cluster	Reload	PPAS service account	Enables warnings if checkpoint segments are filled more frequently than this.	
client_encoding	Session	Immediate	User	Sets the client's character set encoding.	
client_min_messages	Session	Immediate	User	Sets the message levels that are sent to the client.	
commit_delay	Session	Immediate	User	Sets the delay in microseconds between transaction commit and flushing WAL to disk.	

Parameter	Scope of Effect	When Takes Effect	Authorized User	Description	PPAS Only
commit_siblings	Session	Immediate	User	Sets the minimum concurrent open transactions before performing commit_delay.	
config_file	Cluster	Restart	PPAS service account	Sets the server's main configuration file.	
constraint_exclusion	Session	Immediate	User	Enables the planner to use constraints to optimize queries.	
cpu_index_tuple_cost	Session	Immediate	User	Sets the planner's estimate of the cost of processing each index entry during an index scan.	
cpu_operator_cost	Session	Immediate	User	Sets the planner's estimate of the cost of processing each operator or function call.	
cpu_tuple_cost	Session	Immediate	User	Sets the planner's estimate of the cost of processing each tuple (row).	
cursor_tuple_fraction	Session	Immediate	User	Sets the planner's estimate of the fraction of a cursor's rows that will be retrieved.	
custom_variable_classes	Cluster	Reload	PPAS service account	Deprecated in Advanced Server 9.2.	X
data_directory	Cluster	Restart	PPAS service account	Sets the server's data directory.	
DateStyle	Session	Immediate	User	Sets the display format for date and time values.	
db_dialect	Session	Immediate	User	Sets the precedence of built-in namespaces.	X
dbms_alert.max_alerts	Cluster	Restart	PPAS service account	Sets maximum number of alerts.	X
db_user_namespace	Cluster	Reload	PPAS service account	Enables per-database user names.	
deadlock_timeout	Session	Immediate	Superuser	Sets the time to wait on a lock before checking for deadlock.	
debug_assertions	Cluster	Preset	n/a	Turns on various assertion checks. (Not supported in PPAS builds.)	
debug_pretty_print	Session	Immediate	User	Indents parse and plan tree displays.	
debug_print_parse	Session	Immediate	User	Logs each query's parse tree.	
debug_print_plan	Session	Immediate	User	Logs each query's execution plan.	
debug_print_rewritten	Session	Immediate	User	Logs each query's rewritten	

Parameter	Scope of Effect	When Takes Effect	Authorized User	Description	PPAS Only
				parse tree.	
default_heap_fillfactor	Session	Immediate	User	Create new tables with this heap fillfactor by default.	X
default_statistics_target	Session	Immediate	User	Sets the default statistics target.	
default_tablespace	Session	Immediate	User	Sets the default tablespace to create tables and indexes in.	
default_text_search_config	Session	Immediate	User	Sets default text search configuration.	
default_transaction_deferrable	Session	Immediate	User	Sets the default deferrable status of new transactions.	
default_transaction_isolation	Session	Immediate	User	Sets the transaction isolation level of each new transaction.	
default_transaction_read_only	Session	Immediate	User	Sets the default read-only status of new transactions.	
default_with_oids	Session	Immediate	User	Create new tables with OIDs by default.	
default_with_rowids	Session	Immediate	User	Create new tables with ROWID support (OIDs with indexes) by default.	X
dynamic_library_path	Session	Immediate	Superuser	Sets the path for dynamically loadable modules.	
edb_audit	Cluster	Reload	PPAS service account	Enable EDB Auditing to create audit reports in XML or CSV format.	X
edb_audit_connect	Cluster	Reload	PPAS service account	Audits each successful connection.	X
edb_audit_directory	Cluster	Reload	PPAS service account	Sets the destination directory for audit files.	X
edb_audit_disconnect	Cluster	Reload	PPAS service account	Audits end of a session.	X
edb_audit_filename	Cluster	Reload	PPAS service account	Sets the file name pattern for audit files.	X
edb_audit_rotation_day	Cluster	Reload	PPAS service account	Automatic rotation of logfiles based on day of week.	X
edb_audit_rotation_seconds	Cluster	Reload	PPAS service account	Automatic log file rotation will occur after N seconds.	X
edb_audit_rotation_size	Cluster	Reload	PPAS service account	Automatic log file rotation will occur after N Megabytes.	X
edb_audit_statement	Cluster	Reload	PPAS	Sets the type of statements to	X

Parameter	Scope of Effect	When Takes Effect	Authorized User	Description	PPAS Only
			service account	audit.	
edb_connectby_order	Session	Immediate	User	Sort results of CONNECT BY queries with no ORDER BY to depth-first order. **Note: For internal use only**.	X
edb dynatune	Cluster	Restart	PPAS service account	Sets the edb utilization percentage.	X
edb dynatune profile	Cluster	Restart	PPAS service account	Sets the workload profile for dynatune.	X
edb enable icache	Cluster	Restart	PPAS service account	Enable external shared buffer infinitecache mechanism.	X
edb icache compression level	Session	Immediate	Superuser	Sets compression level of infinitecache buffers.	X
edb icache servers	Cluster	Reload	PPAS service account	A list of comma separated *hostname*:*portnumber* icache servers.	X
edb redwood date	Session	Immediate	User	Determines whether DATE should behave like a TIMESTAMP or not.	X
edb redwood strings	Session	Immediate	User	Treat NULL as an empty string when concatenated with a text value.	X
edb sql protect.enabled	Cluster	Reload	PPAS service account	Defines whether SQL/Protect should track queries or not.	X
edb sql protect.level	Cluster	Reload	PPAS service account	Defines the behavior of SQL/Protect when an event is found.	X
edb sql protect.max protected relations	Cluster	Restart	PPAS service account	Sets the maximum number of relations protected by SQL/Protect per role.	X
edb sql protect.max protected roles	Cluster	Restart	PPAS service account	Sets the maximum number of roles protected by SQL/Protect.	X
edb_stmt_level_tx	Session	Immediate	User	Allows continuing on errors instead of requiring a transaction abort.	X
effective cache size	Session	Immediate	User	Sets the planner's assumption about the size of the disk cache.	
effective_io_concurrency	Session	Immediate	User	Number of simultaneous requests that can be handled efficiently by the disk subsystem.	
enable_bitmapscan	Session	Immediate	User	Enables the planner's use of	

Parameter	Scope of Effect	When Takes Effect	Authorized User	Description	PPAS Only
				bitmap-scan plans.	
enable_hashagg	Session	Immediate	User	Enables the planner's use of hashed aggregation plans.	
enable_hashjoin	Session	Immediate	User	Enables the planner's use of hash join plans.	
enable_hints	Session	Immediate	User	Enable optimizer hints in SQL statements.	X
enable_indexonlyscan	Session	Immediate	User	Enables the planner's use of index-only-scan plans.	
enable_indexscan	Session	Immediate	User	Enables the planner's use of index-scan plans.	
enable_material	Session	Immediate	User	Enables the planner's use of materialization.	
enable_mergejoin	Session	Immediate	User	Enables the planner's use of merge join plans.	
enable_nestloop	Session	Immediate	User	Enables the planner's use of nested-loop join plans.	
enable_seqscan	Session	Immediate	User	Enables the planner's use of sequential-scan plans.	
enable_sort	Session	Immediate	User	Enables the planner's use of explicit sort steps.	
enable_tidscan	Session	Immediate	User	Enables the planner's use of TID scan plans.	
escape_string_warning	Session	Immediate	User	Warn about backslash escapes in ordinary string literals.	
event_source	Cluster	Restart	PPAS service account	Sets the application name used to identify PostgreSQL messages in the event log.	
exit_on_error	Session	Immediate	User	Terminate session on any error.	
external_pid_file	Cluster	Restart	PPAS service account	Writes the postmaster PID to the specified file.	
extra_float_digits	Session	Immediate	User	Sets the number of digits displayed for floating-point values.	
from_collapse_limit	Session	Immediate	User	Sets the FROM-list size beyond which subqueries are not collapsed.	
fsync	Cluster	Reload	PPAS service account	Forces synchronization of updates to disk.	
full_page_writes	Cluster	Reload	PPAS service account	Writes full pages to WAL when first modified after a checkpoint.	
geqo	Session	Immediate	User	Enables genetic query optimization.	
geqo_effort	Session	Immediate	User	GEQO: effort is used to set the	

Parameter	Scope of Effect	When Takes Effect	Authorized User	Description	PPAS Only
				default for other GEQO parameters.	
geqo_generations	Session	Immediate	User	GEQO: number of iterations of the algorithm.	
geqo_pool_size	Session	Immediate	User	GEQO: number of individuals in the population.	
geqo_seed	Session	Immediate	User	GEQO: seed for random path selection.	
geqo_selection_bias	Session	Immediate	User	GEQO: selective pressure within the population.	
geqo_threshold	Session	Immediate	User	Sets the threshold of FROM items beyond which GEQO is used.	
gin_fuzzy_search_limit	Session	Immediate	User	Sets the maximum allowed result for exact search by GIN.	
hba_file	Cluster	Restart	PPAS service account	Sets the server's "hba" configuration file.	
hot_standby	Cluster	Restart	PPAS service account	Allows connections and queries during recovery.	
hot_standby_feedback	Cluster	Reload	PPAS service account	Allows feedback from a hot standby to the primary that will avoid query conflicts.	
ident_file	Cluster	Restart	PPAS service account	Sets the server's "ident" configuration file.	
ignore_system_indexes	Cluster/ Session	Reload/ Immediate	PPAS service account/ User	Disables reading from system indexes. (Can also be set with PGOPTIONS at session start.)	
index advisor.enabled	Session	Immediate	User	Enable Index Advisor plugin.	X
integer_datetimes	Cluster	Preset	n/a	Datetimes are integer based.	
IntervalStyle	Session	Immediate	User	Sets the display format for interval values.	
join_collapse_limit	Session	Immediate	User	Sets the FROM-list size beyond which JOIN constructs are not flattened.	
krb_caseins_users	Cluster	Reload	PPAS service account	Sets whether Kerberos and GSSAPI user names should be treated as case-insensitive.	
krb_server_keyfile	Cluster	Reload	PPAS service account	Sets the location of the Kerberos server key file.	
krb_srvname	Cluster	Reload	PPAS service account	Sets the name of the Kerberos service.	
lc_collate	Database	Preset	n/a	Shows the collation order	

Parameter	Scope of Effect	When Takes Effect	Authorized User	Description	PPAS Only
				locale.	
lc_ctype	Database	Preset	n/a	Shows the character classification and case conversion locale.	
lc_messages	Session	Immediate	Superuser	Sets the language in which messages are displayed.	
lc_monetary	Session	Immediate	User	Sets the locale for formatting monetary amounts.	
lc_numeric	Session	Immediate	User	Sets the locale for formatting numbers.	
lc_time	Session	Immediate	User	Sets the locale for formatting date and time values.	
listen_addresses	Cluster	Restart	PPAS service account	Sets the host name or IP address(es) to listen to.	
local_preload_libraries	Cluster/ Session	Reload/ Immediate	PPAS service account/ User	Lists shared libraries to preload into each backend. (Can also be set with PGOPTIONS at session start.)	
lo_compat_privileges	Session	Immediate	Superuser	Enables backward compatibility mode for privilege checks on large objects.	
log_autovacuum_min_duration	Cluster	Reload	PPAS service account	Sets the minimum execution time above which autovacuum actions will be logged.	
log_checkpoints	Cluster	Reload	PPAS service account	Logs each checkpoint.	
log_connections	Cluster/ Session	Reload/ Immediate	PPAS service account/ User	Logs each successful connection. (Can also be set with PGOPTIONS at session start.)	
log_destination	Cluster	Reload	PPAS service account	Sets the destination for server log output.	
log_directory	Cluster	Reload	PPAS service account	Sets the destination directory for log files.	
log_disconnections	Cluster/ Session	Reload/ Immediate	PPAS service account/ User	Logs end of a session, including duration. (Can also be set with PGOPTIONS at session start.)	
log_duration	Session	Immediate	Superuser	Logs the duration of each completed SQL statement.	
log_error_verbosity	Session	Immediate	Superuser	Sets the verbosity of logged messages.	
log_executor_stats	Session	Immediate	Superuser	Writes executor performance statistics to the server log.	

Parameter	Scope of Effect	When Takes Effect	Authorized User	Description	PPAS Only
log_file_mode	Cluster	Reload	PPAS service account	Sets the file permissions for log files.	
log_filename	Cluster	Reload	PPAS service account	Sets the file name pattern for log files.	
log_hostname	Cluster	Reload	PPAS service account	Logs the host name in the connection logs.	
log_line_prefix	Cluster	Reload	PPAS service account	Controls information prefixed to each log line.	
log_lock_waits	Session	Immediate	Superuser	Logs long lock waits.	
log_min_duration_statement	Session	Immediate	Superuser	Sets the minimum execution time above which statements will be logged.	
log_min_error_statement	Session	Immediate	Superuser	Causes all statements generating error at or above this level to be logged.	
log_min_messages	Session	Immediate	Superuser	Sets the message levels that are logged.	
log_parser_stats	Session	Immediate	Superuser	Writes parser performance statistics to the server log.	
log_planner_stats	Session	Immediate	Superuser	Writes planner performance statistics to the server log.	
log_rotation_age	Cluster	Reload	PPAS service account	Automatic log file rotation will occur after N minutes.	
log_rotation_size	Cluster	Reload	PPAS service account	Automatic log file rotation will occur after N kilobytes.	
log_statement	Session	Immediate	Superuser	Sets the type of statements logged.	
log_statement_stats	Session	Immediate	Superuser	Writes cumulative performance statistics to the server log.	
log_temp_files	Session	Immediate	Superuser	Log the use of temporary files larger than this number of kilobytes.	
log_timezone	Cluster	Reload	PPAS service account	Sets the time zone to use in log messages.	
log_truncate_on_rotation	Cluster	Reload	PPAS service account	Truncate existing log files of same name during log rotation.	
logging_collector	Cluster	Restart	PPAS service account	Start a subprocess to capture stderr output and/or csvlogs into log files.	
maintenance_work_mem	Session	Immediate	User	Sets the maximum memory to	

Parameter	Scope of Effect	When Takes Effect	Authorized User	Description	PPAS Only
				be used for maintenance operations.	
max_connections	Cluster	Restart	PPAS service account	Sets the maximum number of concurrent connections.	
max_files_per_process	Cluster	Restart	PPAS service account	Sets the maximum number of simultaneously open files for each server process.	
max_function_args	Cluster	Preset	n/a	Shows the maximum number of function arguments.	
max_identifier_length	Cluster	Preset	n/a	Shows the maximum identifier length.	
max_index_keys	Cluster	Preset	n/a	Shows the maximum number of index keys.	
max_locks_per_transaction	Cluster	Restart	PPAS service account	Sets the maximum number of locks per transaction.	
max_pred_locks_per_transaction	Cluster	Restart	PPAS service account	Sets the maximum number of predicate locks per transaction.	
max_prepared_transactions	Cluster	Restart	PPAS service account	Sets the maximum number of simultaneously prepared transactions.	
max_stack_depth	Session	Immediate	Superuser	Sets the maximum stack depth, in kilobytes.	
max_standby_archive_delay	Cluster	Reload	PPAS service account	Sets the maximum delay before canceling queries when a hot standby server is processing archived WAL data.	
max_standby_streaming_delay	Cluster	Reload	PPAS service account	Sets the maximum delay before canceling queries when a hot standby server is processing streamed WAL data.	
max_wal_senders	Cluster	Restart	PPAS service account	Sets the maximum number of simultaneously running WAL sender processes.	
nls_length_semantics	Session	Immediate	Superuser	Sets the semantics to use for char, varchar, varchar2 and long columns.	X
odbc_lib_path	Cluster	Restart	PPAS service account	Sets the path for ODBC library.	X
optimizer_mode	Session	Immediate	User	Default optimizer mode.	X
oracle_home	Cluster	Restart	PPAS service account	Sets the path for the Oracle home directory.	X
password_encryption	Session	Immediate	User	Encrypt passwords.	
port	Cluster	Restart	PPAS	Sets the TCP port on which the	

Parameter	Scope of Effect	When Takes Effect	Authorized User	Description	PPAS Only
			service account	server listens.	
post_auth_delay	Cluster/ Session	Reload/ Immediate	PPAS service account/ User	Waits N seconds on connection startup after authentication. (Can also be set with PGOPTIONS at session start.)	
pre_auth_delay	Cluster	Reload	PPAS service account	Waits N seconds on connection startup before authentication.	
qreplace_function	Session	Immediate	Superuser	The function to be used by Query Replace feature. **Note: For internal use only**.	X
query_rewrite_enabled	Session	Immediate	User	Child table scans will be skipped if their constraints guarantee that no rows match the query.	X
query_rewrite_integrity	Session	Immediate	Superuser	Sets the degree to which query rewriting must be enforced.	X
quote_all_identifiers	Session	Immediate	User	When generating SQL fragments, quote all identifiers.	
random_page_cost	Session	Immediate	User	Sets the planner's estimate of the cost of a nonsequentially fetched disk page.	
replication_timeout	Cluster	Reload	PPAS service account	Sets the maximum time to wait for WAL replication.	
restart_after_crash	Cluster	Reload	PPAS service account	Reinitialize server after backend crash.	
search_path	Session	Immediate	User	Sets the schema search order for names that are not schema-qualified.	
segment_size	Cluster	Preset	n/a	Shows the number of pages per disk file.	
seq_page_cost	Session	Immediate	User	Sets the planner's estimate of the cost of a sequentially fetched disk page.	
server_encoding	Database	Preset	n/a	Sets the server (database) character set encoding.	
server_version	Cluster	Preset	n/a	Shows the server version.	
server_version_num	Cluster	Preset	n/a	Shows the server version as an integer.	
session_replication_role	Session	Immediate	Superuser	Sets the session's behavior for triggers and rewrite rules.	
shared_buffers	Cluster	Restart	PPAS service account	Sets the number of shared memory buffers used by the server.	
shared_preload_libraries	Cluster	Restart	PPAS	Lists shared libraries to preload	

Parameter	Scope of Effect	When Takes Effect	Authorized User	Description	PPAS Only
			service account	into server.	
sql_inheritance	Session	Immediate	User	Causes subtables to be included by default in various commands.	
ssl	Cluster	Restart	PPAS service account	Enables SSL connections.	
ssl_ca_file	Cluster	Restart	PPAS service account	Location of the SSL certificate authority file.	
ssl_cert_file	Cluster	Restart	PPAS service account	Location of the SSL server certificate file.	
ssl_ciphers	Cluster	Restart	PPAS service account	Sets the list of allowed SSL ciphers.	
ssl_crl_file	Cluster	Restart	PPAS service account	Location of the SSL certificate revocation list file.	
ssl_key_file	Cluster	Restart	PPAS service account	Location of the SSL server private key file.	
ssl_renegotiation_limit	Session	Immediate	User	Set the amount of traffic to send and receive before renegotiating the encryption keys.	
standard_conforming_strings	Session	Immediate	User	Causes '...' strings to treat backslashes literally.	
statement_timeout	Session	Immediate	User	Sets the maximum allowed duration of any statement.	
stats_temp_directory	Cluster	Reload	PPAS service account	Writes temporary statistics files to the specified directory.	
superuser_reserved_connections	Cluster	Restart	PPAS service account	Sets the number of connection slots reserved for superusers.	
synchronize_seqscans	Session	Immediate	User	Enable synchronized sequential scans.	
synchronous_commit	Session	Immediate	User	Sets immediate fsync at commit.	
synchronous_standby_names	Cluster	Reload	PPAS service account	List of names of potential synchronous standbys.	
syslog_facility	Cluster	Reload	PPAS service account	Sets the syslog "facility" to be used when syslog enabled.	
syslog_ident	Cluster	Reload	PPAS	Sets the program name used to	

Parameter	Scope of Effect	When Takes Effect	Authorized User	Description	PPAS Only
			service account	identify PostgreSQL messages in syslog.	
tcp_keepalives_count	Session	Immediate	User	Maximum number of TCP keepalive retransmits.	
tcp_keepalives_idle	Session	Immediate	User	Time between issuing TCP keepalives.	
tcp_keepalives_interval	Session	Immediate	User	Time between TCP keepalive retransmits.	
temp_buffers	Session	Immediate	User	Sets the maximum number of temporary buffers used by each session.	
temp_file_limit	Session	Immediate	Superuser	Limits the total size of all temporary files used by each session.	
temp_tablespaces	Session	Immediate	User	Sets the tablespace(s) to use for temporary tables and sort files.	
timed_statistics	Session	Immediate	User	Enables the recording of timed statistics.	X
timezone	Session	Immediate	User	Sets the time zone for displaying and interpreting time stamps.	
timezone_abbreviations	Session	Immediate	User	Selects a file of time zone abbreviations.	
trace_hints	Session	Immediate	User	Emit debug info about hints being honored.	X
trace_notify	Session	Immediate	User	Generates debugging output for LISTEN and NOTIFY.	
trace_recovery_messages	Cluster	Reload	PPAS service account	Enables logging of recovery-related debugging information.	
trace_sort	Session	Immediate	User	Emit information about resource usage in sorting.	
track_activities	Session	Immediate	Superuser	Collects information about executing commands.	
track_activity_query_size	Cluster	Restart	PPAS service account	Sets the size reserved for pg_stat_activity.current_query, in bytes.	
track_counts	Session	Immediate	Superuser	Collects statistics on database activity.	
track_functions	Session	Immediate	Superuser	Collects function-level statistics on database activity.	
track_io_timing	Session	Immediate	Superuser	Collects timing statistics for database I/O activity.	
transaction_deferrable	Session	Immediate	User	Whether to defer a read-only serializable transaction until it can be executed with no possible serialization failures.	

Parameter	Scope of Effect	When Takes Effect	Authorized User	Description	PPAS Only
transaction_isolation	Session	Immediate	User	Sets the current transaction's isolation level.	
transaction_read_only	Session	Immediate	User	Sets the current transaction's read-only status.	
transform_null_equals	Session	Immediate	User	Treats "expr=NULL" as "expr IS NULL".	
unix_socket_directory	Cluster	Restart	PPAS service account	Sets the directory where the Unix-domain socket will be created.	
unix_socket_group	Cluster	Restart	PPAS service account	Sets the owning group of the Unix-domain socket.	
unix_socket_permissions	Cluster	Restart	PPAS service account	Sets the access permissions of the Unix-domain socket.	
update_process_title	Session	Immediate	Superuser	Updates the process title to show the active SQL command.	
vacuum_cost_delay	Session	Immediate	User	Vacuum cost delay in milliseconds.	
vacuum_cost_limit	Session	Immediate	User	Vacuum cost amount available before napping.	
vacuum_cost_page_dirty	Session	Immediate	User	Vacuum cost for a page dirtied by vacuum.	
vacuum_cost_page_hit	Session	Immediate	User	Vacuum cost for a page found in the buffer cache.	
vacuum_cost_page_miss	Session	Immediate	User	Vacuum cost for a page not found in the buffer cache.	
vacuum_defer_cleanup_age	Cluster	Reload	PPAS service account	Number of transactions by which VACUUM and HOT cleanup should be deferred, if any.	
vacuum_freeze_min_age	Session	Immediate	User	Minimum age at which VACUUM should freeze a table row.	
vacuum_freeze_table_age	Session	Immediate	User	Age at which VACUUM should scan whole table to freeze tuples.	
wal_block_size	Cluster	Preset	n/a	Shows the block size in the write ahead log.	
wal_buffers	Cluster	Restart	PPAS service account	Sets the number of disk-page buffers in shared memory for WAL.	
wal_keep_segments	Cluster	Reload	PPAS service account	Sets the number of WAL files held for standby servers.	
wal_level	Cluster	Restart	PPAS service account	Set the level of information written to the WAL.	

Parameter	Scope of Effect	When Takes Effect	Authorized User	Description	PPAS Only
wal_segment_size	Cluster	Preset	n/a	Shows the number of pages per write ahead log segment.	
wal_sync_method	Cluster	Reload	PPAS service account	Selects the method used for forcing WAL updates to disk.	
wal_writer_delay	Cluster	Reload	PPAS service account	WAL writer sleep time between WAL flushes.	
work_mem	Session	Immediate	User	Sets the maximum memory to be used for query workspaces.	
xmlbinary	Session	Immediate	User	Sets how binary values are to be encoded in XML.	
xmloption	Session	Immediate	User	Sets whether XML data in implicit parsing and serialization operations is to be considered as documents or content fragments.	
zero_damaged_pages	Session	Immediate	Superuser	Continues processing past damaged page headers.	

2.2.3 Configuration Parameters by Functionality

This section provides more detail for certain groups of configuration parameters.

The section heading for each parameter is followed by a list of attributes:

- **Parameter Type.** Type of values the parameter can accept. See Section 2.2.1 for a discussion of parameter type values.
- **Default Value.** Default setting if a value is not explicitly set in the configuration file.
- **Range.** Permitted range of values.
- **Minimum Scope of Effect.** Smallest scope for which a distinct setting can be made. Generally, the minimal scope of a distinct setting is either the entire **cluster** (the setting is the same for all databases and sessions thereof, in the cluster), or **per session** (the setting may vary by role, database, or individual session). (This attribute has the same meaning as the "Scope of Effect" column in the table of Section 2.2.2.)
- **When Value Changes Take Effect.** 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 with a restart of 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 Section 2.2.2.)

- **Required Authorization to Activate.** 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 a PPAS service account (`enterprisedb` or `postgres` depending upon the Advanced Server compatibility installation mode). This attribute has the same meaning as the "Authorized User" column in the table of Section 2.2.2.

2.2.3.1 Top Performance Related Parameters

This section discusses the configuration parameters that have the most immediate impact on performance.

2.2.3.1.1 shared_buffers

Parameter Type: Integer

Default Value: 32MB

Range: 128kB to system dependent

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Restart

Required Authorization to Activate: PPAS service account

Sets the amount of memory the database server uses for shared memory buffers. The default is typically 32 megabytes (`32MB`), but might be less if your kernel settings will not support it (as determined during `initdb`). This setting must be at least 128 kilobytes. (Non-default values of `BLCKSZ` change the minimum.) However, settings significantly higher than the minimum are usually needed 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. There are some workloads where even large settings for `shared_buffers` are effective, but because Postgres Plus also relies on the operating system cache, it is unlikely that an allocation of more than 40% of RAM to `shared_buffers` will work better than a smaller amount. Larger settings for `shared_buffers` usually require a corresponding increase in `checkpoint_segments`, in order to spread out the process of writing large quantities of new or changed data over a longer period of time.

On systems with less than 1GB of RAM, a smaller percentage of RAM is appropriate, so as to leave adequate space for the operating system (15% of memory is more typical in these situations). Also, on Windows, large values for `shared_buffers` aren't as effective. You may find 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 Postgres Plus to request more System V shared memory than your operating system's default configuration allows. See Section 17.4.1, "Shared Memory and Semaphores" in the *PostgreSQL Core Documentation* for information on how to adjust those parameters, if necessary.

2.2.3.1.2 work_mem

Parameter Type: Integer

Default Value: 1MB

Range: 64kB to 2097151kB

Minimum Scope of Effect: Per session

When Value Changes Take Effect: Immediate

Required Authorization to Activate: Session user

Specifies the amount of memory to be used by internal sort operations and hash tables before writing to temporary disk files. The value defaults to one megabyte (`1MB`). Note that for a complex query, several sort or hash operations might be running in parallel; each operation will be allowed to use as much memory as this value specifies before it starts to write data into temporary files. Also, several running sessions could be doing such operations concurrently. Therefore, the total memory used could be many times the value of `work_mem`; it is necessary to keep this fact 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.

2.2.3.1.3 maintenance_work_mem

Parameter Type: Integer

Default Value: 16MB

Range: 1024kB to 2097151kB

Minimum Scope of Effect: Per session

When Value Changes Take Effect: Immediate

Required Authorization to Activate: Session user

Specifies the maximum amount of memory to be used by maintenance operations, such as `VACUUM`, `CREATE INDEX`, and `ALTER TABLE ADD FOREIGN KEY`. It defaults to 16 megabytes (`16MB`). Since only one of these operations can be executed at a time by a database session, 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.

Note that when autovacuum runs, up to `autovacuum_max_workers` times this memory may be allocated, 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 won't necessarily improve performance.

2.2.3.1.4 wal_buffers

Parameter Type: Integer

Default Value: 64kB

Range: 32kB to system dependent

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Restart

Required Authorization to Activate: PPAS service account

The amount of memory used in shared memory for WAL data. The default is 64 kilobytes (`64kB`). The setting need only be large enough to hold the amount of WAL data generated by one typical transaction, since the data is written out to disk at every transaction commit.

Increasing this parameter might cause Postgres Plus to request more System V shared memory than your operating system's default configuration allows. See Section 17.4.1, "Shared Memory and Semaphores" in the *PostgreSQL Core Documentation* for information on how to adjust those parameters, if necessary.

Although even this very small setting does not always cause a problem, there are situations where it can result in extra `fsync` calls, and degrade overall system throughput. Increasing this value to 1MB or so can alleviate this problem. On very busy systems, an even higher value may be needed, up to a maximum of about 16MB. Like `shared_buffers`, this parameter increases Postgres Plus's initial shared memory allocation, so if increasing it causes a Postgres Plus start failure, you will need to increase the operating system limit.

2.2.3.1.5 checkpoint_segments

Parameter Type: Integer

Default Value: 3

Range: 1 to 2147483647

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Reload

Required Authorization to Activate: PPAS service account

Maximum number of log file segments between automatic WAL checkpoints (each segment is normally 16 megabytes). The default is 3. Increasing this parameter uses more disk space and can increase the amount of time needed for crash recovery.

Increasing the `checkpoint_segments` parameter can dramatically improve performance, especially during bulk data loads. A reasonable starting value is 30, even up to 256 for write-heavy systems. Values over 64 are typically used for bulk loads.

Note: Regardless of the setting of `checkpoint_segments`, you will get a checkpoint at least every 5 minutes unless you also increase `checkpoint_timeout` from its default.

2.2.3.1.6 checkpoint_completion_target

Parameter Type: Floating point

Default Value: 0.5

Range: 0 to 1

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Reload

Required Authorization to Activate: PPAS service account

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

The default of 0.5 means aim to finish the checkpoint writes when 50% of the next checkpoint is ready. A value of 0.9 means aim to finish the checkpoint writes when 90% of the next checkpoint is done, thus throttling the checkpoint writes over a larger amount of time and avoiding spikes of performance bottlenecking.

If you have increased `checkpoint_segments`, you should also increase `checkpoint_completion_target` to 0.9; this will decrease the performance impact of checkpointing on a busy system. A large value for `checkpoint_completion_target` is ineffective for small values of `checkpoint_segments`, which is why the default is 0.5.

2.2.3.1.7 checkpoint_timeout

Parameter Type: Integer

Default Value: 5min

Range: 30s to 3600s

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Reload

Required Authorization to Activate: PPAS service account

Maximum time between automatic WAL checkpoints, in seconds. The default is five minutes (`5min`). 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 the aforementioned adjustments to the checkpoint parameters is that your system will use a modest amount of additional disk space, and will take longer to recover in the event of a crash. However, for most users, this is a small price to pay for a significant performance improvement.

2.2.3.1.8 bgwriter_delay

Parameter Type: Integer

Default Value: 200ms

Range: 10ms to 10000ms

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Reload

Required Authorization to Activate: PPAS 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 (controllable by the `bgwriter_lru_maxpages` and `bgwriter_lru_multiplier` parameters). It then sleeps for `bgwriter_delay` milliseconds, and repeats.

The default value is 200 milliseconds (`200ms`). Note that on many systems, the effective resolution of sleep delays is 10 milliseconds; setting `bgwriter_delay` to a value that is not a multiple of 10 might have the same results as setting it to the next higher multiple of 10.

Typically, when tuning `bgwriter_delay`, it should be reduced from its default value. This parameter is rarely increased, except perhaps to save on power consumption on a system with very low utilization.

2.2.3.1.9 seq_page_cost

Parameter Type: Floating point

Default Value: 1

Range: 0 to 1.79769e+308

Minimum Scope of Effect: Per session

When Value Changes Take Effect: Immediate

Required Authorization to Activate: Session user

Sets the planner's estimate of the cost of a disk page fetch that is part of a series of sequential fetches. The default is 1.0. This value can be overridden for a particular

tablespace by setting the tablespace parameter of the same name. (Refer to the `ALTER TABLESPACE` command in the *PostgreSQL Core Documentation*.)

The default value assumes very little caching, so it's frequently 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 less than 1 (rather than its default value of 1) to see whether you get better query plans that way. If your database fits entirely within memory, you can lower this value much more, perhaps to 0.1.

2.2.3.1.10 random_page_cost

Parameter Type: Floating point

Default Value: 4

Range: 0 to 1.79769e+308

Minimum Scope of Effect: Per session

When Value Changes Take Effect: Immediate

Required Authorization to Activate: Session user

Sets the planner's estimate of the cost of a non-sequentially-fetched disk page. The default is 4.0. This value can be overridden for a particular tablespace by setting the tablespace parameter of the same name. (Refer to the `ALTER TABLESPACE` command in the *PostgreSQL Core Documentation*.)

Reducing this value relative to `seq_page_cost` will cause the system to prefer index scans; raising it will make 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 frequently 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 (rather than its default of 4) to see whether you get better query plans that way. If your database fits entirely within memory, you can lower this value much more, perhaps to 0.1.

Although the system will let you do so, 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 within memory, since in that case there is no penalty for touching pages out of sequence. Also, in a heavily-cached database you should lower

both values relative to the CPU parameters, since the cost of fetching a page already in RAM is much smaller than it would normally be.

2.2.3.1.11 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 is available to a single query. This is factored into estimates of the cost of using an index; a higher value makes it more likely index scans will be used, a lower value makes it more likely sequential scans will be used. When setting this parameter you should consider both Postgres Plus's shared buffers and the portion of the kernel's disk cache that will be used for Postgres Plus data files. Also, take into account the expected number of concurrent queries on different tables, since they will have to share the available space. This parameter has no effect on the size of shared memory allocated by Postgres Plus, nor does it reserve kernel disk cache; it is used only for estimation purposes. The default is 128 megabytes (`128MB`).

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

2.2.3.1.12 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 transaction commit will wait for WAL records to be written to disk before the command returns a "success" indication to the client. The default, and safe, setting is on. 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, but the database state will be just the same as if those transactions had been 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 Section 29.3, "Asynchronous Commit" in the *PostgreSQL Core Documentation* for information.

This parameter can be changed at any time; the behavior for any one transaction is determined by the setting in effect when it commits. It is 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` within the transaction.

2.2.3.2 Resource Consumption / Memory

The configuration parameters in this section control resource usage pertaining to memory.

2.2.3.2.1 edb_dynatune

Parameter Type: Integer

Default Value: 0

Range: 0 to 100

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Restart

Required Authorization to Activate: PPAS service account

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 Postgres Plus 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.

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 Postgres Plus 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 Postgres Plus 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.

2.2.3.2.2 edb_dynatune_profile

Parameter Type: Enum

Default Value: `oltp`

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

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Restart

Required Authorization to Activate: PPAS service account

This parameter is used to control tuning aspects based upon 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.

2.2.3.2.3 edb_enable_icache

Parameter Type: Boolean

Default Value: `false`

Range: {`true` | `false`}

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Restart

Required Authorization to Activate: PPAS service account

Enables or disables Infinite Cache. If `edb_enable_icache` is set to on, Infinite Cache is enabled; if the parameter is set to off, Infinite Cache is disabled.

If you set `edb_enable_icache` to on, you must also specify a list of cache servers by setting the `edb_icache_servers` parameter.

2.2.3.2.4 edb_icache_servers

Parameter Type: String

Default Value: none

Range: n/a

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Reload

Required Authorization to Activate: PPAS service account

The `edb_icache_servers` parameter specifies a list of one or more servers with active edb-icache daemons. `edb_icache_servers` is a string value that takes the form of a comma-separated list of *`hostname`*`:`*`port`* pairs. You can specify each pair in any of the following forms:

- *`hostname`*
- *`IP_address`*
- *`hostname`*`:`*`portnumber`*
- *`IP_address`*`:`*`portnumber`*

If you do not specify a port number, Infinite Cache assumes that the cache server is listening at port 11211. This configuration parameter will take effect only if `edb_enable_icache` is set to on. Use the `edb_icache_servers` parameter to specify a maximum of 128 cache nodes.

You can dynamically modify the Infinite Cache server nodes. To change the Infinite Cache server configuration, use the `edb_icache_servers` parameter in the `postgresql.conf` file to perform the following:

- Specify additional cache information to add server(s).
- Delete server information to remove server(s).
- Specify additional server information and delete existing server information to both add and delete servers during the same reload operation.

After updating the `edb_icache_servers` parameter in the `postgresql.conf` file, you must reload the configuration parameters for the changes to take effect.

2.2.3.2.5 edb_icache_compression_level

Parameter Type: Integer

Default Value: 6

Range: 0 to 9

Minimum Scope of Effect: Per session

When Value Changes Take Effect: Immediate

Required Authorization to Activate: Superuser

The `edb_icache_compression_level` parameter controls the compression level that is applied to each page before storing it in the distributed Infinite Cache.

When Advanced Server reads data from disk, it typically reads the data in 8kB increments. If `edb_icache_compression_level` is set to 0, each time Advanced Server sends an 8kB page to the Infinite Cache server that page is stored (uncompressed) in 8kB of cache memory. If the `edb_icache_compression_level` parameter is set to 9, Advanced Server applies the maximum compression possible before sending it to the Infinite Cache server, so a page that previously took 8kB of cached memory might take 2kB of cached memory. Exact compression numbers are difficult to predict, as they are dependent on the nature of the data on each page.

This parameter must be an integer in the range 0 to 9.

- A compression level of 0 disables compression; it uses no CPU time for compression, but requires more storage space and network resources to process.
- A compression level of 9 invokes the maximum amount of compression; it increases the load on the CPU, but less data flows across the network, so network demand is reduced. Each page takes less room on the Infinite Cache server, so memory requirements are reduced.
- A compression level of 5 or 6 is a reasonable compromise between the amount of compression received and the amount of CPU time invested.

The compression level must be set by the superuser and can be changed for the current session while the server is running. The following command disables the compression mechanism for the currently active session:

```
SET edb_icache_compression_level TO 0;
```

2.2.3.3 Query Planning / Optimizer Hints

This section describes the configuration parameters used for optimizer hints.

2.2.3.3.1 enable_hints

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

Optimizer hints embedded in SQL commands are utilized when `enable_hints` is on. Optimizer hints are ignored when this parameter is off.

2.2.3.3.2 optimizer_mode

Parameter Type: Enum

Default Value: `choose`

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

Minimum Scope of Effect: Per session

When Value Changes Take Effect: Immediate

Required Authorization to Activate: Session user

Sets the default optimization mode for analyzing optimizer hints.

The following table shows the possible values:

Table 2 - Optimizer Modes

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

These optimization modes are based upon the assumption that the client submitting the SQL command is interested in viewing only the first "n" rows of the result set and will then abandon the remainder of the result set. Resources allocated to the query are adjusted as such.

2.2.3.4 Error Reporting and Logging / What to Log

The configuration parameters in this section control error reporting and logging.

2.2.3.4.1 trace_hints

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

Use with the optimizer hints feature to provide more detailed information regarding whether or not a hint was used by the planner. Set the `client_min_messages` and `trace_hints` configuration parameters as follows:

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

The `SELECT` command with the `NO_INDEX` hint shown below illustrates the additional information produced when the aforementioned configuration parameters are set.

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

2.2.3.5 Advanced Server Auditing Settings

This section describes configuration parameters used by the Postgres Plus Advanced Server database auditing feature.

2.2.3.5.1 edb_audit

Parameter Type: Enum

Default Value: `none`

Range: {`none` | `csv` | `xml`}

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Reload

Required Authorization to Activate: PPAS service account

Enables or disables database auditing. The values `xml` or `csv` will enable database auditing. These values represent the file format in which auditing information will be captured. `none` will disable database auditing and is also the default.

2.2.3.5.2 edb_audit_directory

Parameter Type: String

Default Value: `edb_audit`

Range: n/a

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Reload

Required Authorization to Activate: PPAS service account

Specifies the directory where the audit log files will be created. The path of the directory can be absolute or relative to the *`POSTGRES_PLUS_HOME`*`/data` directory.

2.2.3.5.3 edb_audit_filename

Parameter Type: String

Default Value: `audit-%Y%m%d_%H%M%S`

Range: n/a

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Reload

Required Authorization to Activate: PPAS service account

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

2.2.3.5.4 edb_audit_rotation_day

Parameter Type: String

Default Value: `every`

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

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Reload

Required Authorization to Activate: PPAS 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.

2.2.3.5.5 edb_audit_rotation_size

Parameter Type: Integer

Default Value: 0MB

Range: 0MB to 5000MB

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Reload

Required Authorization to Activate: PPAS service account

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

2.2.3.5.6 edb_audit_rotation_seconds

Parameter Type: Integer

Default Value: 0s

Range: 0s to 2147483647s

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Reload

Required Authorization to Activate: PPAS service account

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

2.2.3.5.7 edb_audit_connect

Parameter Type: Enum

Default Value: `failed`

Range: {`none` | `failed` | `all`}

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Reload

Required Authorization to Activate: PPAS 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`.

2.2.3.5.8 edb_audit_disconnect

Parameter Type: Enum

Default Value: `none`

Range: {`none` | `all`}

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Reload

Required Authorization to Activate: PPAS 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`.

2.2.3.5.9 edb_audit_statement

Parameter Type: String

Default Value: `ddl, error`

Range: {`none` | `ddl` | `dml` | `select` | `error` | `all`} ...

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Reload

Required Authorization to Activate: PPAS service account

This configuration parameter is used to specify auditing of different categories of SQL statements. To audit statements resulting in error, set the parameter value to `error`. To audit DDL statements such as `CREATE TABLE`, `ALTER TABLE`, etc., set the parameter value to `ddl`. Modification statements such as `INSERT`, `UPDATE`, `DELETE` etc., can be audited by setting `edb_audit_statement` to `dml`. Setting the value to `all` will audit every statement while `none` disables this feature.

2.2.3.6 Client Connection Defaults / Statement Behavior

This section describes configuration parameters affecting statement behavior.

2.2.3.6.1 default_heap_fillfactor

Parameter Type: Integer

Default Value: 100

Range: 10 to 100

Minimum Scope of Effect: Per session

When Value Changes Take Effect: Immediate

Required Authorization to Activate: Session user

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

The fillfactor for a table is a percentage between 10 and 100. 100 (complete packing) is the default. When a smaller fillfactor is specified, `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 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, but in heavily updated tables smaller fillfactors are appropriate.

2.2.3.7 Client Connection Defaults / Other Defaults

The parameters in this section set other miscellaneous client connection defaults.

2.2.3.7.1 oracle_home

Parameter Type: String

Default Value: none

Range: n/a

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Restart

Required Authorization to Activate: PPAS service account

Before creating an Oracle Call Interface (OCI) database link to an Oracle server, you must direct Advanced Server to the correct Oracle home directory. You can either set the `LD_LIBRARY_PATH` environment variable (or `PATH` on Windows) to the `lib` directory of the Oracle client installation directory or set the value of the `oracle_home` configuration parameter in the `postgresql.conf` file. The value specified in the `oracle_home` configuration parameter will override the `LD_LIBRARY_PATH` (or `PATH` on Windows) environment variable.

The `LD_LIBRARY_PATH` (or `PATH` on Windows) environment variable must be set properly each time you start Advanced Server. To set the `oracle_home` configuration parameter in the `postgresql.conf` file, edit the file, adding the following line:

```
oracle_home = 'lib_directory'
```

Substitute the name of the directory that contains `libclntsh.so` (on Linux) or `oci.dll` (on Windows) for *`lib_directory`*.

2.2.3.7.2 odbc_lib_path

Parameter Type: String

Default Value: none

Range: n/a

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Restart

Required Authorization to Activate: PPAS service account

If you will be using an ODBC driver manager, and if it is installed in a non-standard location, you 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 (typically `libodbc.so`).

2.2.3.8 Version and Platform Compatibility / Oracle Compatibility

The configuration parameters described in this section control various Oracle compatibility features.

2.2.3.8.1 edb_redwood_date

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 `DATE` appears as the data type of a column in the commands, it is translated to `TIMESTAMP(0)` at the time the table definition is stored in the database if the configuration parameter `edb_redwood_date` is set to `TRUE`. Thus, a time component will also be stored in the column along with the date. This is consistent with Oracle's `DATE` data type.

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 such as the data type of a variable in an SPL declaration section, or the

data type of a formal parameter in an SPL procedure or SPL function, or the return type of an SPL function, it is always internally translated to a `TIMESTAMP(0)` and thus, can handle a time component if present.

2.2.3.8.2 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

In Oracle, when a string is concatenated with a null variable or null column, the result is the original string; however, in PostgreSQL concatenation of a string with a null variable or null column gives a null result. If the `edb_redwood_strings` parameter is set to `TRUE`, the aforementioned concatenation operation results in the original string as done by Oracle. If `edb_redwood_strings` is set to `FALSE`, the native PostgreSQL behavior is maintained.

The following example illustrates the difference.

The sample application contains a table of employees. This table has a column named `comm` that is null for most employees. The following query is run with `edb_redwood_string` set to `FALSE`. The concatenation of 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;

      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. The concatenation of an empty string with a non-empty string produces the non-empty string. This result is consistent with the results produced by Oracle for the same query.

```
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;

      EMPLOYEE COMPENSATION
--------------------------------
 SMITH          800.00
 ALLEN        1,600.00     300.00
 WARD         1,250.00     500.00
 JONES        2,975.00
 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)
```

2.2.3.8.3 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

In Oracle, when a runtime error occurs in a SQL command, all the updates on the database caused by that single command are rolled back. This is called *statement level transaction isolation*. 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 have not yet been committed or rolled back are pending until a `COMMIT` or `ROLLBACK` command is executed.

In Postgres Plus, 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 be started.

If `edb_stmt_level_tx` is set to `TRUE`, then an exception will not automatically roll back prior uncommitted database updates, emulating the Oracle behavior. If `edb_stmt_level_tx` is set to `FALSE`, then an exception will roll back uncommitted database updates.

Note: Use `edb_stmt_level_tx` set to `TRUE` only when absolutely necessary, as this may cause a negative performance impact.

The following 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. Note that in PSQL, the command `\set AUTOCOMMIT off` must be issued, otherwise every statement commits automatically defeating the purpose of this demonstration of 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".

COMMIT;
SELECT empno, ename, deptno FROM emp WHERE empno > 9000;

empno | ename | deptno
-------+-------+--------
(0 rows)
```

In the following example, with `edb_stmt_level_tx` set to `TRUE`, the first `INSERT` command has not been 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;
```

```
empno | ename | deptno
-------+-------+--------
  9001 | JONES |     40
(1 row)

COMMIT;
```

A `ROLLBACK` command could have been issued instead of the `COMMIT` command in which case the insert of employee number `9001` would have been rolled back as well.

2.2.3.8.4 db_dialect

Parameter Type: Enum

Default Value: `postgres`

Range: {`postgres` | `redwood`}

Minimum Scope of Effect: Per session

When Value Changes Take Effect: Immediate

Required Authorization to Activate: Session user

Sets the precedence of built-in namespaces in accordance to whether PostgreSQL or Oracle is the preferred compatibility behavior.

In addition to the native PostgreSQL system catalog, `pg_catalog`, Advanced Server provides system catalogs that are compatible with Oracle and Microsoft® SQL Server®. These are `sys` for Oracle and `dbo` for SQL Server. 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`, `sys`, then `dbo`, giving the PostgreSQL catalog the highest precedence. When set to `redwood`, the namespace precedence is `sys`, `dbo`, then `pg_catalog`., giving the Oracle catalog the highest precedence.

2.2.3.8.5 default_with_rowids

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 set to on, `CREATE TABLE` includes a `ROWID` column in newly created tables, which can then be referenced in Oracle compatible SQL commands.

2.2.3.9 Customized Options

In previous releases of Advanced Server, the `custom_variable_classes` was required by those parameters not normally known to be added by add-on modules (such as procedural languages).

2.2.3.9.1 custom_variable_classes

The `custom_variable_classes` parameter is deprecated in Advanced Server 9.2; parameters that previously depended on this parameter no longer require its support.

2.2.3.9.2 dbms_alert.max_alerts

Parameter Type: Integer

Default Value: 100

Range: 0 to 500

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Restart

Required Authorization to Activate: PPAS service account

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

2.2.3.9.3 index_advisor.enabled

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

Provides the capability to temporarily suspend Index Advisor in an EDB-PSQL or PSQL session. The Index Advisor plugin, `index_advisor`, must be loaded in the EDB-PSQL or PSQL session in order to use the `index_advisor.enabled` configuration parameter.

The Index Advisor plugin can be loaded as shown by the following example:

```
$ psql -d edb -U enterprisedb
Password for user enterprisedb:
psql (9.2.0.0)
Type "help" for help.

edb=# LOAD 'index_advisor';
LOAD
```

Use the `SET` command to change the parameter setting to control whether or not Index Advisor generates an alternative query plan as shown by the following example:

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

2.2.3.9.4 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: PPAS service account

Controls whether or not 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 are ready to begin monitoring with SQL/Protect set this parameter to on.

2.2.3.9.5 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: PPAS service account

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

The `edb_sql_protect.level` configuration parameter can be set to one of the following values to use either learn mode, passive mode, or active mode:

- **learn.** Tracks the activities of protected roles and records the relations used by the roles. This is used 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 does not stop any SQL statements from executing. This is the next step after SQL/Protect has learned the expected behavior of the protected roles. This essentially behaves in intrusion detection mode and can be run in production when properly monitored.
- **active.** Stops all invalid statements for a protected role. This behaves as a SQL firewall preventing 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, but it tracks the blocked queries allowing administrators to be alerted before the attacker finds an alternate method of penetrating the system.

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

2.2.3.9.6 edb_sql_protect.max_protected_relations

Parameter Type: Integer

Default Value: 1024

Range: 1 to 2147483647

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Restart

Required Authorization to Activate: PPAS service account

Sets the maximum number of relations that can be protected per role.

2.2.3.9.7 edb_sql_protect.max_protected_roles

Parameter Type: Integer

Default Value: 64

Range: 1 to 2147483647

Minimum Scope of Effect: Cluster

When Value Changes Take Effect: Restart

Required Authorization to Activate: PPAS service account

Sets the maximum number of roles that can be protected.

2.2.3.10 Ungrouped

Configuration parameters in this section apply to Postgres Plus Advanced Server only and are for a specific, limited purpose.

2.2.3.10.1 nls_length_semantics

Parameter Type: Enum

Default Value: `byte`

Range: {byte | char}

Minimum Scope of Effect: Per session

When Value Changes Take Effect: Immediate

Required Authorization to Activate: Superuser

This parameter is provided for Oracle syntax compatibility only and has no effect in Postgres Plus Advanced Server.

For example, the following Oracle form of the ALTER SESSION command is accepted in Advanced Server without throwing a syntax error, but does not alter the session environment:

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

Note: Since the setting of this parameter has no effect on the server environment, it does not appear in the system view pg_settings.

2.2.3.10.2 query_rewrite_enabled

Parameter Type: Enum

Default Value: false

Range: {true | false | force}

Minimum Scope of Effect: Per session

When Value Changes Take Effect: Immediate

Required Authorization to Activate: Session user

This parameter is provided for Oracle syntax compatibility only and has no effect in Postgres Plus Advanced Server.

For example, the following Oracle form of the ALTER SESSION command is accepted in Advanced Server without throwing a syntax error, but does not alter the session environment:

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

Note: Since the setting of this parameter has no effect on the server environment, it does not appear in the system view pg_settings.

2.2.3.10.3 query_rewrite_integrity

Parameter Type: Enum

Default Value: `enforced`

Range: {`enforced` | `trusted` | `stale_tolerated`}

Minimum Scope of Effect: Per session

When Value Changes Take Effect: Immediate

Required Authorization to Activate: Superuser

This parameter is provided for Oracle syntax compatibility only and has no effect in Postgres Plus Advanced Server.

For example, the following Oracle form of the `ALTER SESSION` command is accepted in Advanced Server without throwing a syntax error, but does not alter the session environment:

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

Note: Since the setting of this parameter has no effect on the server environment, it does not appear in the system view `pg_settings`.

2.2.3.10.4 timed_statistics

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

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

Note: When Advanced Server is installed, the `postgresql.conf` file contains an explicit entry setting `timed_statistics` to off. If this entry is commented out letting

timed_statistics to default, and the configuration file is reloaded, timed statistics collection would be turned on.

2.3 Controlling the Audit Logs

Postgres Plus Advanced Server allows database and security administrators, auditors, and operators to track and analyze database activities using the audit logs. The audit logs can be configured to information such as:

- When a role establishes a connection to an Advanced Server database.
- What database objects a role creates, modifies or deletes when he or she is connected to Advanced Server.
- When any failed authentication attempts occur.

You can use parameters specified in the postgresql.conf file to control the information included in the audit logs.

2.3.1 Auditing Configuration Parameters

Use the following postgresql.conf configuration parameters to control database auditing:

edb_audit

> Enables or disables database auditing. The values xml or csv will enable database auditing. These values represent the file format in which auditing information will be captured. none will disable database auditing and is also the default. This option can only be set at server start or in the postgresql.conf file.

edb_audit_directory

> Specifies the directory where the log files will be created. The path of the directory can be relative or absolute to the data folder. This option can only be set at server start or in the postgresql.conf configuration file.

edb_audit_filename

> Specifies the file name of the audit file where the auditing information will be stored. The default file name will be audit-%Y%m%d_%H%M%S. The escape sequences, %Y, %m etc., will be replaced by the appropriate current values according to the system date and time. This option can only be set at server start or in the postgresql.conf configuration file.

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. This option can only be set at server start or in the postgresql.conf configuration file.

edb_audit_rotation_size

Specifies a file size threshold in megabytes when file rotation will be 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 will not occur. This option can only be set at server start or in the postgresql.conf configuration file.

edb_audit_rotation_seconds

Specifies the rotation time in seconds when a new log file should be created. To disable this feature, set this parameter to 0. This option can only be set at server start or in the postgresql.conf configuration file.

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 failed connection attempts, set the value to failed. To audit all connection attempts, set the value to all. This option can only be set at server start or in the postgresql.conf configuration file.

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. This option can only be set at server start or in the postgresql.conf configuration file.

edb_audit_statement

This configuration parameter is used to specify auditing of different categories of SQL statements. To audit statements resulting in error, set the parameter value to error. To audit DDL statements such as CREATE TABLE, ALTER TABLE, etc., set the parameter value to ddl. Modification statements such as INSERT, UPDATE, DELETE etc., can be audited by setting edb_audit_statement to dml. Setting the value to all will audit every statement while none disables this

feature. This option can only be set at server start or in the `postgresql.conf` configuration file.

The following steps describe how to configure Advanced Server to log all connections, disconnections, DDL statements, and any statements resulting in an error. The resulting audit file will rotate every Sunday.

1. Enable auditing by the setting the `edb_audit` parameter to `xml` or `csv`.
2. Set the file rotation day when the new file will be created by setting the parameter `edb_audit_rotation_day` to `sun`.
3. To audit all connections, set the parameter, `edb_audit_connect`, to `all`.
4. To audit all disconnections, set the parameter, `edb_audit_disconnect`, to `all`.
5. To audit all DDL statements and error statements, set the parameter, `edb_audit_statement`, to `dll, error`.

The following is the CSV and XML output when auditing is enabled:

CSV Audit Log File

```
,,,1452,,,2008-03-13 12:40:02 ,startup,"AUDIT:  database system is ready"
enterprisedb,mgmtsvr,127.0.0.1(1266),1620,47d9595b.654,0,2008-03-13 12:42:03 ,connect,"AUDIT:
connection authorized: user=enterprisedb database=mgmtsvr"
enterprisedb,mgmtsvr,127.0.0.1(1266),1620,47d9595b.654,1588,2008-03-13 12:42:08 ,ddl,"AUDIT:
statement: drop table HILOSEQUENCES
     "
enterprisedb,mgmtsvr,127.0.0.1(1266),1620,47d9595b.654,1590,2008-03-13 12:42:09 ,ddl,"AUDIT:
statement: create table HILOSEQUENCES (
                SEQUENCENAME varchar(50) not null,
                HIGHVALUES integer not null,
                constraint hilo_pk primary key (SEQUENCENAME)
            )
     "
enterprisedb,edb,127.0.0.1(1269),904,47d9598d.388,0,2008-03-13 12:42:53 ,connect,"AUDIT:
connection authorized: user=enterprisedb database=edb"
enterprisedb,edb,127.0.0.1(1269),904,47d9598d.388,1618,2008-03-13 12:43:02 ,ddl,"AUDIT:
statement: CREATE TABLE test (f1 INTEGER);"
enterprisedb,edb,127.0.0.1(1269),904,47d9598d.388,1620,2008-03-13 12:43:02 ,sql
statement,"AUDIT:  statement: SELECT * FROM testx;"
enterprisedb,edb,127.0.0.1(1269),904,47d9598d.388,1620,2008-03-13 12:43:02 ,error,"ERROR:
relation "testx" does not exist"
enterprisedb,edb,127.0.0.1(1269),904,47d9598d.388,1621,2008-03-13 12:43:04 ,ddl,"AUDIT:
statement: DROP TABLE test;"
enterprisedb,edb,127.0.0.1(1269),904,47d9598d.388,0,2008-03-13 12:43:20 ,disconnect,"AUDIT:
disconnection: session time: 0:00:26.953 user=enterprisedb database=edb host=127.0.0.1
port=1269"
enterprisedb,mgmtsvr,127.0.0.1(1266),1620,47d9595b.654,0,2008-03-13 12:43:29
,disconnect,"AUDIT:  disconnection: session time: 0:01:26.594 user=enterprisedb
database=mgmtsvr host=127.0.0.1 port=1266"
,,,3148,,,2008-03-13 12:43:35 ,shutdown,"AUDIT:  database system is shut down"
```

XML Audit Log File

```
     <event process_id="2516" time="2008-03-13 13:22:42 " type="startup">
            <message>AUDIT:  database system is ready</message>
     </event>
     <event user="enterprisedb" database="mgmtsvr" remote_host_and_port="127.0.0.1(1281)"
            process_id="352" session_id="47d96338.160" transaction="0"
```

```
        time="2008-03-13 13:24:08 " type="connect">
        <message>AUDIT:  connection authorized: user=enterprisedb
                database=mgmtsvr</message>
</event>
<event user="enterprisedb" database="mgmtsvr" remote_host_and_port="127.0.0.1(1281)"
       process_id="352" session_id="47d96338.160" transaction="1635"
       time="2008-03-13 13:24:10 " type="ddl">
       <command>AUDIT:  statement: drop table HILOSEQUENCES</command>
</event>
<event user="enterprisedb" database="mgmtsvr" remote_host_and_port="127.0.0.1(1281)"
       process_id="352" session_id="47d96338.160" transaction="1637"
       time="2008-03-13 13:24:10 " type="ddl">
       <command>AUDIT:  statement: create table HILOSEQUENCES (
           SEQUENCENAME varchar(50) not null,
           HIGHVALUES integer not null,
           constraint hilo_pk primary key (SEQUENCENAME)
        )</command>
</event>
<event user="enterprisedb" database="edb" remote_host_and_port="127.0.0.1(1283)"
       process_id="3776" session_id="47d96378.ec0" transaction="0"
       time="2008-03-13 13:25:12 " type="connect">
       <message>AUDIT:  connection authorized: user=enterprisedb database=edb</message>
</event>
<event user="enterprisedb" database="edb" remote_host_and_port="127.0.0.1(1283)"
       process_id="3776" session_id="47d96378.ec0" transaction="1667"
       time="2008-03-13 13:25:17 " type="ddl">
       <command>AUDIT:  statement: CREATE TABLE test (f1 INTEGER);</command>
</event>
<event user="enterprisedb" database="edb" remote_host_and_port="127.0.0.1(1283)"
       process_id="3776" session_id="47d96378.ec0" transaction="1669"
       time="2008-03-13 13:25:17 " type="sql statement">
       <command>AUDIT:  statement: SELECT * FROM testx;</command>
</event>
<event user="enterprisedb" database="edb" remote_host_and_port="127.0.0.1(1283)"
       process_id="3776" session_id="47d96378.ec0" transaction="1669"
       time="2008-03-13 13:25:17 " type="error">
       <message>ERROR:  relation &amp;quot;testx&amp;quot; does not exist</message>
</event>
<event user="enterprisedb" database="edb" remote_host_and_port="127.0.0.1(1283)"
       process_id="3776" session_id="47d96378.ec0" transaction="1670"
       time="2008-03-13 13:25:18 " type="ddl">
       <command>AUDIT:  statement: DROP TABLE test;</command>
</event>
<event user="enterprisedb" database="edb" remote_host_and_port="127.0.0.1(1283)"
       process_id="3776" session_id="47d96378.ec0" transaction="0"
       time="2008-03-13 13:25:22 " type="disconnect">
       <message>AUDIT:  disconnection: session time: 0:00:10.094 user=enterprisedb
                database=edb host=127.0.0.1 port=1283</message>
</event>
<event user="enterprisedb" database="mgmtsvr" remote_host_and_port="127.0.0.1(1281)"
       process_id="352" session_id="47d96338.160" transaction="0"
       time="2008-03-13 13:25:31 " type="disconnect">
       <message>AUDIT:  disconnection: session time: 0:01:23.046 user=enterprisedb
                database=mgmtsvr host=127.0.0.1 port=1281</message>
</event>
<event process_id="2768" time="2008-03-13 13:25:36 " type="shutdown">
       <message>AUDIT:  database system is shut down</message>
</event>
```

3 Protecting Against SQL Injection Attacks

Postgres Plus Advanced Server provides protection against SQL injection attacks. A *SQL injection attack* is an attempt to compromise a database by running SQL statements whose results provide clues to the attacker as to the content, structure, or security of that database.

Preventing a SQL injection attack is normally the responsibility of the application developer. The database administrator typically has little or no control over the potential threat. The difficulty for database administrators is that the application must have access to the data to function properly.

SQL/Protect is a module that allows a database administrator to protect a database from SQL injection attacks. SQL/Protect provides a layer of security in addition to the normal database security policies by examining incoming queries for common SQL injection profiles.

SQL/Protect gives the control back to the database administrator by alerting the administrator to potentially dangerous queries and by blocking these queries.

3.1 SQL/Protect Overview

This section contains an introduction to the different types of SQL injection attacks and describes how SQL/Protect guards against them.

3.1.1 Types of SQL Injection Attacks

There are a number of different techniques used to perpetrate SQL injection attacks. Each technique is characterized by a certain *signature*. SQL/Protect examines queries for the following signatures:

Unauthorized Relations

While Postgres Plus Advanced Server allows administrators to restrict access to relations (tables, views, etc.), many administrators do not perform this tedious task. SQL/Protect provides a *learn* mode that tracks the relations a user accesses.

This allows administrators to examine the workload of an application, and for SQL/Protect to learn which relations an application should be allowed to access for a given user or group of users in a role.

When SQL/Protect is switched to either *passive* or *active* mode, the incoming queries are checked against the list of learned relations.

Utility Commands

A common technique used in SQL injection attacks is to run utility commands, which are typically SQL Data Definition Language (DDL) statements. An example is creating a user-defined function that has the ability to access other system resources.

SQL/Protect can block the running of all utility commands, which are not normally needed during standard application processing.

SQL Tautology

The most frequent technique used in SQL injection attacks is issuing a tautological `WHERE` clause condition (that is, using a condition that is always true).

The following is an example:

```
WHERE password = 'x' OR 'x'='x'
```

Attackers will usually start identifying security weaknesses using this technique. SQL/Protect can block queries that use a tautological conditional clause.

Unbounded DML Statements

A dangerous action taken during SQL injection attacks is the running of unbounded DML statements. These are `UPDATE` and `DELETE` statements with no `WHERE` clause. For example, an attacker may update all users' passwords to a known value or initiate a denial of service attack by deleting all of the data in a key table.

3.1.1.1 Protected Roles

Monitoring for SQL injection attacks involves analyzing SQL statements originating in database sessions where the current user of the session is a protected role. A *protected role* is a Postgres Plus Advanced Server user or group that the database administrator has chosen to monitor using SQL/Protect. (In Postgres Plus Advanced Server, users and groups are collectively referred to as *roles*.)

Each protected role can be customized for the types of SQL injection attacks (discussed in Section 3.1.1) for which it is to be monitored, thus providing different levels of protection by role and significantly reducing the user maintenance load for DBAs.

Note: A role with the superuser privilege cannot be made a protected role. If a protected non-superuser role is subsequently altered to become a superuser, certain behaviors are exhibited whenever an attempt is made by that superuser to issue any command:

- A warning message is issued by SQL/Protect on every command issued by the protected superuser.

- The statistic in column `superusers` of `edb_sql_protect_stats` is incremented with every command issued by the protected superuser. See Step 3 of Section 3.2.2.2 for information on the `edb_sql_protect_stats` view.
- When SQL/Protect is in active mode, all commands issued by the protected superuser are prevented from running.

A protected role that has the superuser privilege should either be altered so that it is no longer a superuser, or it should be reverted back to an unprotected role.

3.1.1.2 Attack Attempt Statistics

Each usage of a command by a protected role that is considered an attack by SQL/Protect is recorded. These statistics are accessible from a view that can be easily monitored to identify the start of a potential attack.

If a role is protected in more than one database, the role's statistics for attacks in each database are maintained separately and are viewable only when connected to the respective database.

Statistics are collected by type of SQL injection attack as discussed in Section 3.1.1.

This gives database administrators the opportunity to react proactively in preventing theft of valuable data or other malicious actions.

Note: SQL/Protect statistics are maintained in memory while the database server is running. When the database server is shut down, the statistics are saved to a binary file named `edb_sqlprotect.stat` in the `data/global` subdirectory of the Postgres Plus Advanced Server home directory.

3.2 Configuring SQL/Protect

The library file (`sqlprotect.so` on Linux, `sqlprotect.dll` on Windows) necessary to run SQL/Protect should already be installed in the `lib` subdirectory of your Postgres Plus Advanced Server home directory.

You will also need the SQL script file `sqlprotect.sql` located in the `share/contrib` subdirectory of your Postgres Plus Advanced Server home directory.

You must configure the database server to use SQL/Protect, and you must configure each database that you want SQL/Protect to monitor:

- The database server configuration file, `postgresql.conf`, must be modified by adding and enabling configuration parameters used by SQL/Protect.
- Database objects used by SQL/Protect must be installed in each database that you want SQL/Protect to monitor.

Step 1: Edit the following configuration parameters in the `postgresql.conf` file located in the `data` subdirectory of your Postgres Plus Advanced Server home directory.

- **shared_preload_libraries.** Add `$libdir/sqlprotect` to the list of libraries.
- **edb_sql_protect.enabled.** Controls whether or not 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 are ready to begin monitoring with SQL/Protect set this parameter to `on`. If this parameter is omitted, the default is `off`.
- **edb_sql_protect.level.** Sets the action taken by SQL/Protect when a SQL statement is issued by a protected role. If this parameter is omitted, the default behavior is `passive`. Initially, set this parameter to `learn`. See Section 3.2.1.2 for further explanation of this parameter.
- **edb_sql_protect.max_protected_roles.** Sets the maximum number of roles that can be protected. If this parameter is omitted, the default setting is `64`.
- **edb_sql_protect.max_protected_relations.** Sets the maximum number of relations that can be protected per role. If this parameter is omitted, the default setting is `1024`.

The following example shows the settings of these parameters in the `postgresql.conf` file:

```
shared_preload_libraries =
'$libdir/dbms_pipe,$libdir/edb_gen,$libdir/plugins/plugin_debugger,$libdir/plugins/plugin_spl_
debugger,$libdir/sqlprotect'
                                        # (change requires restart)
                    .
                    .
                    .
edb_sql_protect.enabled = off
edb_sql_protect.level = learn
edb_sql_protect.max_protected_roles = 64
edb_sql_protect.max_protected_relations = 1024
```

Step 2: Restart the database server after you have modified the `postgresql.conf` file.

For Linux only: Run the `/etc/init.d/ppas-9.2` script with the `restart` option as shown by the following:

```
$ su root
Password:
$ /etc/init.d/ppas-9.2 restart
Stopping Postgres Plus Advanced Server 9.2:
waiting for server to shut down.... done
server stopped
Starting Postgres Plus Advanced Server 9.2:
waiting for server to start.... done
server started
Postgres Plus Advanced Server 9.2 started successfully
```

For Windows only: Open Control Panel, Administrative Tools, and then Services. Restart the service named `ppas-9.2`.

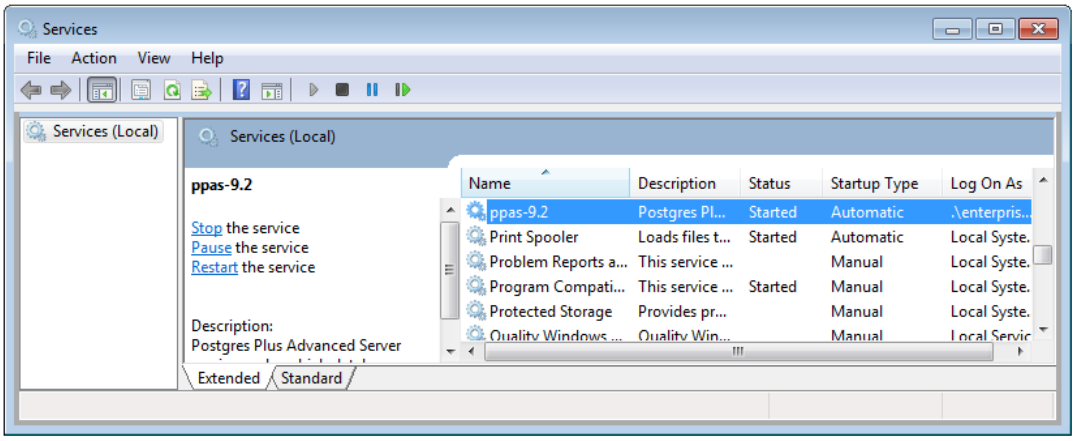

Figure 1 - ppas-9.2 service

Step 3: For each database that you want to protect from SQL injection attacks, connect to the database as a superuser (either `enterprisedb` or `postgres`, depending upon your installation options) and run the script `sqlprotect.sql` located in the `share/contrib` subdirectory of your Postgres Plus Advanced Server home directory.

The script creates the SQL/Protect database objects in a schema named `sqlprotect`.

The following example shows this process to set up protection for a database named `edb`:

```
$ /opt/PostgresPlus/9.2AS/bin/psql -d edb -U enterprisedb
Password for user enterprisedb:
psql (9.2.0.0)
Type "help" for help.

edb=# \i /opt/PostgresPlus/9.2AS/share/contrib/sqlprotect.sql
CREATE SCHEMA
GRANT
SET
CREATE TABLE
GRANT
CREATE TABLE
GRANT
CREATE FUNCTION
CREATE FUNCTION
CREATE FUNCTION
CREATE FUNCTION
CREATE FUNCTION
CREATE FUNCTION
CREATE FUNCTION
CREATE FUNCTION
CREATE VIEW
GRANT
CREATE VIEW
GRANT
CREATE VIEW
```

```
GRANT
CREATE FUNCTION
CREATE FUNCTION
SET
```

3.2.1 Selecting Roles to Protect

After the SQL/Protect database objects have been created in a database, you select the roles for which SQL queries are to be monitored for protection, and the level of protection.

3.2.1.1 Setting the Protected Roles List

For each database that you want to protect, you must determine the roles you want to monitor and then add those roles to the *protected roles list* of that database.

Step 1: Connect as a superuser to a database that you wish to protect using either `psql` or Postgres Enterprise Manager Client.

```
$ /opt/PostgresPlus/9.2AS/bin/psql -d edb -U enterprisedb
Password for user enterprisedb:
psql (9.2.0.0)
Type "help" for help.

edb=#
```

Step 2: Since the SQL/Protect tables, functions, and views are built under the `sqlprotect` schema, use the `SET search_path` command to include the `sqlprotect` schema in your search path. This eliminates the need to schema-qualify any operation or query involving SQL/Protect database objects.

```
edb=# SET search_path TO sqlprotect;
SET
```

Step 3: Each role that you wish to protect must be added to the protected roles list. This list is maintained in the table `edb_sql_protect`.

To add a role, use the function `protect_role('rolename')`.

The following example protects a role named `appuser`.

```
edb=# SELECT protect_role('appuser');
 protect_role
--------------

(1 row)
```

You can list the roles that have been added to the protected roles list by is suing the following query:

```
edb=# SELECT * FROM edb sql protect;
 dbid  | roleid | protect_relations | allow_utility_cmds | allow_tautology | allow_empty_dml
-------+--------+-------------------+--------------------+-----------------+-----------------
 13917 |  16671 | t                 | f                  | f               | f
(1 row)
```

A view is also provided that gives the same information using the object names instead of the Object Identification numbers (OIDs).

```
edb=# \x
Expanded display is on.
edb=# SELECT * FROM list_protected_users;
-[ RECORD 1 ]------+--------
dbname             | edb
username           | appuser
protect_relations  | t
allow_utility_cmds | f
allow_tautology    | f
allow_empty_dml    | f
```

3.2.1.2 Setting the Protection Level

Configuration parameter `edb_sql_protect.level` sets the protection level, which defines the behavior of SQL/Protect when a protected role issues a SQL statement. **The defined behavior applies to all roles in the protected roles lists of all databases configured with SQL/Protect in the database server.**

In the `postgresql.conf` file the `edb_sql_protect.level` configuration parameter can be set to one of the following values to use either learn mode, passive mode, or active mode:

- **learn.** Tracks the activities of protected roles and records the relations used by the roles. This is used 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 does not stop any SQL statements from executing. This is the next step after SQL/Protect has learned the expected behavior of the protected roles. This essentially behaves in intrusion detection mode and can be run in production when properly monitored.
- **active.** Stops all invalid statements for a protected role. This behaves as a SQL firewall preventing 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, but it tracks the blocked queries allowing administrators to be alerted before the attacker finds an alternate method of penetrating the system.

If the `edb_sql_protect.level` parameter is not set or is omitted from the configuration file, the default behavior of SQL/Protect is passive.

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

3.2.2 Monitoring Protected Roles

Once you have configured SQL/Protect in a database, added roles to the protected roles list, and set the desired protection level, you can then activate SQL/Protect in one of learn mode, passive mode, or active mode. You can then start running your applications.

With a new SQL/Protect installation, the first step is to determine the relations that protected roles should be permitted to access during normal operation. Learn mode allows a role to run applications during which time SQL/Protect is recording the relations that are accessed. These are added to the role's *protected relations list* stored in table `edb_sql_protect_rel`.

Monitoring for protection against attack begins when SQL/Protect is run in passive or active mode. In passive and active modes, the role is permitted to access the relations in its protected relations list as these were determined to be the relations the role should be able to access during typical usage.

However, if a role attempts to access a relation that is not in its protected relations list, a `WARNING` or `ERROR` severity level message is returned by SQL/Protect. The role's attempted action on the relation may or may not be carried out depending upon whether the mode is passive or active.

3.2.2.1 Learn Mode

Step 1: To activate SQL/Protect in learn mode, set the following parameters in the `postgresql.conf` file as shown below:

```
edb_sql_protect.enabled = on
edb_sql_protect.level = learn
```

Step 2: Reload the `postgresql.conf` file.

Choose Expert Configuration, then Reload Configuration from the Postgres Plus Advanced Server application menu.

Note: For an alternative method of reloading the configuration file, use the `pg_reload_conf` function. Be sure you are connected to a database as a superuser and execute function `pg_reload_conf` as shown by the following example:

```
edb=# SELECT pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Step 3: Allow the protected roles to run their applications.

As an example the following queries are issued in the `psql` application by protected role `appuser`:

```
edb=> SELECT * FROM dept;
NOTICE:  SQLPROTECT: Learned relation: 16384
 deptno |   dname    |   loc
--------+------------+----------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
(4 rows)

edb=> SELECT empno, ename, job FROM emp WHERE deptno = 10;
NOTICE:  SQLPROTECT: Learned relation: 16391
 empno | ename  |    job
-------+--------+-----------
  7782 | CLARK  | MANAGER
  7839 | KING   | PRESIDENT
  7934 | MILLER | CLERK
(3 rows)
```

SQL/Protect generates a `NOTICE` severity level message indicating the relation has been added to the role's protected relations list.

In SQL/Protect learn mode, SQL statements that are cause for suspicion are not prevented from executing, but a message is issued to alert the user to potentially dangerous statements as shown by the following example:

```
edb=> CREATE TABLE appuser_tab (f1 INTEGER);
NOTICE:  SQLPROTECT: This command type is illegal for this user
CREATE TABLE
edb=> DELETE FROM appuser_tab;
NOTICE:  SQLPROTECT: Learned relation: 16672
NOTICE:  SQLPROTECT: Illegal Query: empty DML
DELETE 0
```

Step 4: As a protected role runs applications, the SQL/Protect tables can be queried to observe the addition of relations to the role's protected relations list.

Connect as a superuser to the database you are monitoring and set the search path to include the `sqlprotect` schema.

```
edb=# SET search_path TO sqlprotect;
SET
```

Query the `edb_sql_protect_rel` table to see the relations added to the protected relations list:

```
edb=# SELECT * FROM edb_sql_protect_rel;
 dbid  | roleid | relid
-------+--------+-------
```

```
 13917 |  16671 | 16384
 13917 |  16671 | 16391
 13917 |  16671 | 16672
(3 rows)
```

The view `list_protected_rels` is provided that gives more comprehensive information along with the object names instead of the OIDs.

```
edb=# SELECT * FROM list_protected_rels;
 Database | Protected User | Schema |    Name     | Type  |    Owner
----------+----------------+--------+-------------+-------+-------------
 edb      | appuser        | public | dept        | Table | enterprisedb
 edb      | appuser        | public | emp         | Table | enterprisedb
 edb      | appuser        | public | appuser_tab | Table | appuser
(3 rows)
```

3.2.2.2 Passive Mode

Once you have determined that a role's applications have accessed all relations they will need, you can now change the protection level so that SQL/Protect can actively monitor the incoming SQL queries and protect against SQL injection attacks.

Passive mode is the less restrictive of the two protection modes, passive and active.

Step 1: To activate SQL/Protect in passive mode, set the following parameters in the `postgresql.conf` file as shown below:

```
edb_sql_protect.enabled = on
edb_sql_protect.level = passive
```

Step 2: Reload the configuration file as shown in Step 2 of Section 3.2.2.1.

Now SQL/Protect is in passive mode. For relations that have been learned such as the `dept` and `emp` tables of the prior examples, SQL statements are permitted with no special notification to the client by SQL/Protect as shown by the following queries run by user `appuser`:

```
edb=> SELECT * FROM dept;
 deptno |   dname    |   loc
--------+------------+----------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
(4 rows)

edb=> SELECT empno, ename, job FROM emp WHERE deptno = 10;
 empno | ename  |    job
-------+--------+-----------
  7782 | CLARK  | MANAGER
  7839 | KING   | PRESIDENT
  7934 | MILLER | CLERK
(3 rows)
```

SQL/Protect does not prevent any SQL statement from executing, but issues a message of WARNING severity level for SQL statements executed against relations that were not learned, or for SQL statements that contain a prohibited signature as shown in the following example:

```
edb=> CREATE TABLE appuser_tab_2 (f1 INTEGER);
WARNING:  SQLPROTECT: This command type is illegal for this user
CREATE TABLE
edb=> INSERT INTO appuser_tab_2 VALUES (1);
WARNING:  SQLPROTECT: Illegal Query: relations
INSERT 0 1
edb=> INSERT INTO appuser_tab_2 VALUES (2);
WARNING:  SQLPROTECT: Illegal Query: relations
INSERT 0 1
edb=> SELECT * FROM appuser_tab_2 WHERE 'x' = 'x';
WARNING:  SQLPROTECT: Illegal Query: relations
WARNING:  SQLPROTECT: Illegal Query: tautology
 f1
----
  1
  2
(2 rows)
```

Step 3: Monitor the statistics for suspicious activity.

By querying the view edb_sql_protect_stats, you can see the number of times SQL statements were executed that referenced relations that were not in a role's protected relations list, or contained SQL injection attack signatures.

The columns in edb_sql_protect_stats monitor the following:

- **username.** Name of the protected role.
- **superusers.** Number of SQL statements issued when the protected role is a superuser. In effect, any SQL statement issued by a protected superuser increases this statistic. See Section 3.1.1.1 for information on protected superusers.
- **relations.** Number of SQL statements issued referencing relations that were not learned by a protected role. (That is, relations that are not in a role's protected relations list.)
- **commands.** Number of DDL statements issued by a protected role.
- **tautology.** Number of SQL statements issued by a protected role that contained a tautological condition.
- **dml.** Number of UPDATE and DELETE statements issued by a protected role that did not contain a WHERE clause.

The following is a query on edb_sql_protect_stats:

```
edb=# SET search_path TO sqlprotect;
SET
```

```
edb=# SELECT * FROM edb_sql_protect_stats;
 username | superusers | relations | commands | tautology | dml
----------+------------+-----------+----------+-----------+-----
 appuser  |          0 |         3 |        1 |         1 |   0
(1 row)
```

3.2.2.3 Active Mode

In active mode, disallowed SQL statements are prevented from executing. Also, the message issued by SQL/Protect has a higher severity level of `ERROR` instead of `WARNING`.

Step 1: To activate SQL/Protect in active mode, set the following parameters in the `postgresql.conf` file as shown below:

```
edb_sql_protect.enabled = on
edb_sql_protect.level = active
```

Step 2: Reload the configuration file as shown in Step 2 of Section 3.2.2.1.

The following example illustrates SQL statements similar to those given in the examples of Step 2 in Section 3.2.2.2, but executed by user `appuser` when `edb_sql_protect.level` is set to `active`:

```
edb=> CREATE TABLE appuser_tab_3 (f1 INTEGER);
ERROR:  SQLPROTECT: This command type is illegal for this user
edb=> INSERT INTO appuser_tab_2 VALUES (1);
ERROR:  SQLPROTECT: Illegal Query: relations
edb=> SELECT * FROM appuser_tab_2 WHERE 'x' = 'x';
ERROR:  SQLPROTECT: Illegal Query: relations
```

The following shows the resulting statistics:

```
edb=# SELECT * FROM sqlprotect.edb_sql_protect_stats;
 username | superusers | relations | commands | tautology | dml
----------+------------+-----------+----------+-----------+-----
 appuser  |          0 |         5 |        2 |         1 |   0
(1 row)
```

3.3 Common Maintenance Operations

The following describes how to perform other common operations.

You must be connected as a superuser to perform these operations and have included schema `sqlprotect` in your search path.

3.3.1 Adding a Role to the Protected Roles List

To add a role to the protected roles list run `protect_role('rolename')`.

```
protect_role('rolename')
```

This is shown by the following example:

```
edb=# SELECT protect_role('newuser');
 protect_role
--------------

(1 row)
```

3.3.2 Removing a Role From the Protected Roles List

To remove a role from the protected roles list use either of the following functions:

```
unprotect_role('rolename')
unprotect_role(roleoid)
```

Note: The variation of the function using the OID is useful if you remove the role using the `DROP ROLE` or `DROP USER` SQL statement before removing the role from the protected roles list. If a query on a SQL/Protect relation returns a value such as `unknown (OID=16458)` for the user name, use the `unprotect_role(roleoid)` form of the function to remove the entry for the deleted role from the protected roles list.

Removing a role using these functions also removes the role's protected relations list.

The statistics for a role that has been removed are not deleted until you use the `drop_stats` function as described in Section 3.3.5.

The following is an example of the `unprotect_role` function:

```
edb=# SELECT unprotect_role('newuser');
 unprotect_role
----------------

(1 row)
```

Alternatively, the role could be removed by giving its OID of `16693`:

```
edb=# SELECT unprotect_role(16693);
 unprotect_role
----------------

(1 row)
```

3.3.3 Setting the Types of Protection for a Role

You can change whether or not a role is protected from a certain type of SQL injection attack.

Change the Boolean value for the column in `edb_sql_protect` corresponding to the type of SQL injection attack for which protection of a role is to be disabled or enabled.

Be sure to qualify the following columns in your `WHERE` clause of the statement that updates `edb_sql_protect`:

- **dbid.** OID of the database for which you are making the change
- **roleid.** OID of the role for which you are changing the Boolean settings

For example, to allow a given role to issue utility commands, update the `allow_utility_cmds` column as follows:

```
UPDATE edb_sql_protect SET allow_utility_cmds = TRUE WHERE dbid = 13917 AND
roleid = 16671;
```

You can verify the change was made by querying `edb_sql_protect` or `list_protected_users`. In the following query note that column `allow_utility_cmds` now contains `t`.

```
edb=# SELECT dbid, roleid, allow_utility_cmds FROM edb_sql_protect;
 dbid  | roleid | allow_utility_cmds
-------+--------+--------------------
 13917 |  16671 | t
(1 row)
```

The updated rules take effect on new sessions started by the role since the change was made.

3.3.4 Removing a Relation From the Protected Relations List

If SQL/Protect has learned that a given relation is accessible for a given role, you can subsequently remove that relation from the role's protected relations list.

Delete its entry from the `edb_sql_protect_rel` table using any of the following functions:

```
unprotect_rel('rolename', 'relname')
unprotect_rel('rolename', 'schema', 'relname')
unprotect_rel(roleoid, reloid)
```

If the relation given by *`relname`* is not in your current search path, specify the relation's schema using the second function format.

The third function format allows you to specify the OIDs of the role and relation, respectively, instead of their text names.

The following example illustrates the removal of the `public.emp` relation from the protected relations list of the role `appuser`.

```
edb=# SELECT unprotect_rel('appuser', 'public', 'emp');
 unprotect_rel
```

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

(1 row)
```

The following query shows there is no longer an entry for the `emp` relation.

```
edb=# SELECT * FROM list_protected_rels;
 Database | Protected User | Schema |    Name     | Type  |    Owner
----------+----------------+--------+-------------+-------+-------------
 edb      | appuser        | public | dept        | Table | enterprisedb
 edb      | appuser        | public | appuser_tab | Table | appuser
(2 rows)
```

SQL/Protect will now issue a warning or completely block access (depending upon the setting of `edb_sql_protect.level`) whenever the role attempts to utilize that relation.

3.3.5 Deleting Statistics

You can delete statistics from view `edb_sql_protect_stats` using either of the two following functions:

```
drop_stats('rolename')
drop_stats(roleoid)
```

Note: The variation of the function using the OID is useful if you remove the role using the `DROP ROLE` or `DROP USER` SQL statement before deleting the role's statistics using `drop_stats('rolename')`. If a query on `edb_sql_protect_stats` returns a value such as `unknown (OID=16458)` for the user name, use the `drop_stats(roleoid)` form of the function to remove the deleted role's statistics from `edb_sql_protect_stats`.

The following is an example of the `drop_stats` function:

```
edb=# SELECT drop_stats('appuser');
 drop_stats
------------

(1 row)

edb=# SELECT * FROM edb_sql_protect_stats;
 username | superusers | relations | commands | tautology | dml
----------+------------+-----------+----------+-----------+-----
(0 rows)
```

The following is an example of using the `drop_stats(roleoid)` form of the function when a role is dropped before deleting its statistics:

```
edb=# SELECT * FROM edb_sql_protect_stats;
      username       | superusers | relations | commands | tautology | dml
---------------------+------------+-----------+----------+-----------+-----
 unknown (OID=16693) |          0 |         5 |        3 |         1 |   0
```

```
 appuser              |          0 |         5 |        2 |         1 |   0
(2 rows)

edb=# SELECT drop_stats(16693);
 drop_stats
------------

(1 row)

edb=# SELECT * FROM edb_sql_protect_stats;
 username | superusers | relations | commands | tautology | dml
----------+------------+-----------+----------+-----------+-----
 appuser  |          0 |         5 |        2 |         1 |   0
(1 row)
```

3.3.6 Disabling and Enabling Monitoring

If you wish to turn off SQL/Protect monitoring once you have enabled it, perform the following steps:

Step 1: Set the configuration parameter `edb_sql_protect.enabled` to `off` in the `postgresql.conf` file.

The entry for `edb_sql_protect.enabled` should look like the following:

```
edb_sql_protect.enabled = off
```

Step 2: Reload the configuration file as shown in Step 2 of Section 3.2.2.1.

To re-enable SQL/Protect monitoring perform the following steps:

Step 1: Set the configuration parameter `edb_sql_protect.enabled` to `on` in the `postgresql.conf` file.

The entry for `edb_sql_protect.enabled` should look like the following:

```
edb_sql_protect.enabled = on
```

Step 2: Reload the configuration file as shown in Step 2 of Section 3.2.2.1.

3.4 Backing Up and Restoring a SQL/Protect Database

Backing up a database that is configured with SQL/Protect, and then restoring the backup file to a new database require additional considerations to what is normally associated with backup and restore procedures. This is primarily due to the use of Object Identification numbers (OIDs) in the SQL/Protect tables as explained in this section.

Note: This section is applicable if your backup and restore procedures result in the re-creation of database objects in the new database with new OIDs such as is the case when using the `pg_dump` backup program.

If you are backing up your Postgres Plus Advanced Server database server by simply using the operating system's copy utility to create a binary image of the Postgres Plus Advanced Server data files (file system backup method), then this section does not apply.

3.4.1 Object Identification Numbers in SQL/Protect Tables

SQL/Protect uses two tables, `edb_sql_protect` and `edb_sql_protect_rel`, to store information on database objects such as databases, roles, and relations. References to these database objects in these tables are done using the objects' OIDs, and not the objects' text names. The OID is a numeric data type used by Postgres Plus Advanced Server to uniquely identify each database object.

When a database object is created, Postgres Plus Advanced Server assigns an OID to the object, which is then used whenever a reference is needed to the object in the database catalogs. If you create the same database object in two databases, such as a table with the same `CREATE TABLE` statement, each table is assigned a different OID in each database.

In a backup and restore operation that results in the re-creation of the backed up database objects, the restored objects end up with different OIDs in the new database than what they were assigned in the original database. As a result, the OIDs referencing databases, roles, and relations stored in the `edb_sql_protect` and `edb_sql_protect_rel` tables are no longer valid when these tables are simply dumped to a backup file and then restored to a new database.

The following sections describe two functions, `export_sqlprotect` and `import_sqlprotect`, that are used specifically for backing up and restoring SQL/Protect tables in order to ensure the OIDs in the SQL/Protect tables reference the correct database objects after the SQL/Protect tables are restored.

3.4.2 Backing Up the Database

The following are the steps to back up a database that has been configured with SQL/Protect.

Step 1: Create a backup file using `pg_dump`.

The following example shows a plain-text backup file named `/tmp/edb.dmp` created from database `edb` using the `pg_dump` utility program:

```
$ cd /opt/PostgresPlus/9.2AS/bin
$ ./pg_dump -U enterprisedb -Fp -f /tmp/edb.dmp edb
Password:
$
```

Step 2: Connect to the database as a superuser and export the SQL/Protect data using the `export_sqlprotect('sqlprotect_file')` function where *`sqlprotect_file`* is the fully qualified path to a file where the SQL/Protect data is to be saved.

The `enterprisedb` operating system account (`postgres` if you installed Postgres Plus Advanced Server in PostgreSQL compatibility mode) must have read and write access to the directory specified in *`sqlprotect_file`*.

```
edb=# SELECT sqlprotect.export_sqlprotect('/tmp/sqlprotect.dmp');
 export_sqlprotect
-------------------

(1 row)
```

The files `/tmp/edb.dmp` and `/tmp/sqlprotect.dmp` comprise your total database backup.

3.4.3 Restoring From the Backup Files

Step 1: Restore the backup file to the new database.

The following example uses the `psql` utility program to restore the plain-text backup file `/tmp/edb.dmp` to a newly created database named `newdb`:

```
$ /opt/PostgresPlus/9.2AS/bin/psql -d newdb -U enterprisedb -f /tmp/edb.dmp
Password for user enterprisedb:
SET
SET
SET
SET
SET
COMMENT
CREATE SCHEMA
    .
    .
    .
```

Step 2: Connect to the new database as a superuser and delete all rows from the `edb_sql_protect_rel` table.

This step removes any existing rows in the `edb_sql_protect_rel` table that were backed up from the original database. These rows do not contain the correct OIDs relative to the database where the backup file has been restored.

```
$ /opt/PostgresPlus/9.2AS/bin/psql -d newdb -U enterprisedb
Password for user enterprisedb:
psql (9.2.0.0)
Type "help" for help.

newdb=# DELETE FROM sqlprotect.edb_sql_protect_rel;
DELETE 2
```

Step 3: Delete all rows from the `edb_sql_protect` table.

This step removes any existing rows in the `edb_sql_protect` table that were backed up from the original database. These rows do not contain the correct OIDs relative to the database where the backup file has been restored.

```
newdb=# DELETE FROM sqlprotect.edb_sql_protect;
DELETE 1
```

Step 4: Delete any statistics that may exist for the database.

This step removes any existing statistics that may exist for the database to which you are restoring the backup. The following query displays any existing statistics:

```
newdb=# SELECT * FROM sqlprotect.edb_sql_protect_stats;
 username | superusers | relations | commands | tautology | dml
----------+------------+-----------+----------+-----------+-----
(0 rows)
```

For each row that appears in the preceding query, use the `drop_stats` function specifying the role name of the entry.

For example, if a row appeared with `appuser` in the `username` column, issue the following command to remove it:

```
newdb=# SELECT sqlprotect.drop_stats('appuser');
 drop_stats
------------

(1 row)
```

Step 5: Make sure the role names that were protected by SQL/Protect in the original database exist in the database server where the new database resides.

If the original and new databases reside in the same database server, then nothing needs to be done assuming you have not deleted any of these roles from the database server.

Step 6: Run the function `import_sqlprotect('`*`sqlprotect_file`*`')` where *`sqlprotect_file`* is the fully qualified path to the file you created in Step 2 of Section 3.4.2.

```
newdb=# SELECT sqlprotect.import_sqlprotect('/tmp/sqlprotect.dmp');
 import_sqlprotect
-------------------

(1 row)
```

Tables `edb_sql_protect` and `edb_sql_protect_rel` are now populated with entries containing the OIDs of the database objects as assigned in the new database. The

statistics view `edb_sql_protect_stats` also now displays the statistics imported from the original database.

The SQL/Protect tables and statistics are now properly restored for this database. This is verified by the following queries on the Postgres Plus Advanced Server system catalogs:

```
newdb=# SELECT datname, oid FROM pg database;
  datname  |  oid
-----------+-------
 template1 |     1
 template0 | 13909
 edb       | 13917
 newdb     | 16679
(4 rows)

newdb=# SELECT rolname, oid FROM pg_roles;
   rolname    |  oid
--------------+-------
 enterprisedb |    10
 appuser      | 16671
 newuser      | 16678
(3 rows)

newdb=# SELECT relname, oid FROM pg_class WHERE relname IN ('dept','emp','appuser_tab');
   relname   |  oid
-------------+-------
 appuser_tab | 16803
 dept        | 16809
 emp         | 16812
(3 rows)

newdb=# SELECT * FROM sqlprotect.edb_sql_protect;
 dbid  | roleid | protect_relations | allow_utility_cmds | allow_tautology | allow_empty_dml
-------+--------+-------------------+--------------------+-----------------+-----------------
 16679 |  16671 | t                 | t                  | f               | f
(1 row)

newdb=# SELECT * FROM sqlprotect.edb sql protect rel;
 dbid  | roleid | relid
-------+--------+-------
 16679 |  16671 | 16809
 16679 |  16671 | 16803
(2 rows)

newdb=# SELECT * FROM sqlprotect.edb_sql_protect_stats;
 username | superusers | relations | commands | tautology | dml
----------+------------+-----------+----------+-----------+-----
 appuser  |          0 |         5 |        2 |         1 |   0
(1 row)
```

Note the following about the columns in tables `edb_sql_protect` and `edb_sql_protect_rel`:

- **dbid.** Matches the value in the `oid` column from `pg_database` for `newdb`
- **roleid.** Matches the value in the `oid` column from `pg_roles` for `appuser`

Also note that in table `edb_sql_protect_rel`, the values in the `relid` column match the values in the `oid` column of `pg_class` for relations `dept` and `appuser_tab`.

Step 7: Verify that the SQL/Protect configuration parameters are set as desired in the `postgresql.conf` file for the database server running the new database as described throughout sections 3.2, 3.2.1, and 3.2.2. Restart the database server or reload the configuration file as appropriate.

You can now monitor the database using SQL/Protect.

4 Application Development

This chapter describes various application programming interfaces and other development tools.

4.1 libpq C Library

libpq is the C application programmer's interface to Postgres Plus Advanced Server. libpq is a set of library functions that allow client programs to pass queries to the Postgres Plus Advanced Server and to receive the results of these queries.

libpq is also the underlying engine for several other EnterpriseDB application interfaces including those written for C++, Perl, Python, Tcl and ECPG. So some aspects of libpq's behavior will be 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.

4.1.1 Using libpq with EnterpriseDB SPL

The EnterpriseDB 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

4.1.2 REFCURSOR Support

In earlier releases, 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.

Please note that the samples that follow do not include error-handling code that would be required in a real-world client application.

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);

  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);

  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 will contain 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
```

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 will contain 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;
  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 will return 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
  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
```

4.1.3 Array Binding

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

4.1.3.1 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`.

API Definition

```
PGresult * PQBulkStart(PGconn *conn,
                       const char * Stmt_Name,
                       unsigned int nCol,
                       const int *paramFmts);
```

4.1.3.2 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.

API Definition

```
PGresult *PQexecBulk(PGconn *conn,
                     unsigned int nRows,
                     const char *const * paramValues,
                     const int *paramLengths);
```

4.1.3.3 PQBulkFinish

This function completes the current bulk operation. You can use the prepared statement again without re-preparing it.

API Definition

```
PGresult *PQBulkFinish(PGconn *conn);
```

4.1.3.4 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 will be used repeatedly will be parsed and planned just once, rather than each time they are executed.

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);
```

4.1.3.5 Example Code (Using PQBulkStart, PQexecBulk, PQBulkFinish)

The following example uses `PGBulkStart`, `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" );
}
```

4.1.3.6 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 );
}
```

4.2 Debugger

The Debugger is a tool that gives developers and DBAs the ability to test and debug Postgres Plus 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.

The Debugger is integrated with and invoked from Postgres Enterprise Manager Client. There are two basic ways the Debugger can be used to test programs:

- **Standalone Debugging.** 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-Context Debugging.** 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 Section 4.2.5.3.

4.2.1 Configuring the Debugger

Before using the Debugger, you must edit the `postgresql.conf` file located in the `data` subdirectory of your Postgres Plus Advanced Server home directory by adding `$libdir/plugin_debugger` to the list of libraries in configuration parameter `shared_preload_libraries` as shown by the following:

```
shared_preload_libraries = '$libdir/dbms_pipe,$libdir/edb_gen,$libdir/plugin_debugger'
```

Restart the database server after making this modification.

4.2.2 Starting the Debugger

You can start the Debugger for standalone debugging in either of the following two ways.

- In the Postgres Enterprise Manager Client Object Browser, highlight the stored procedure or function you wish to debug. From the main menu bar, select the Tools menu, choose Debugging from the menu, then Debug from the submenu as shown by the following:

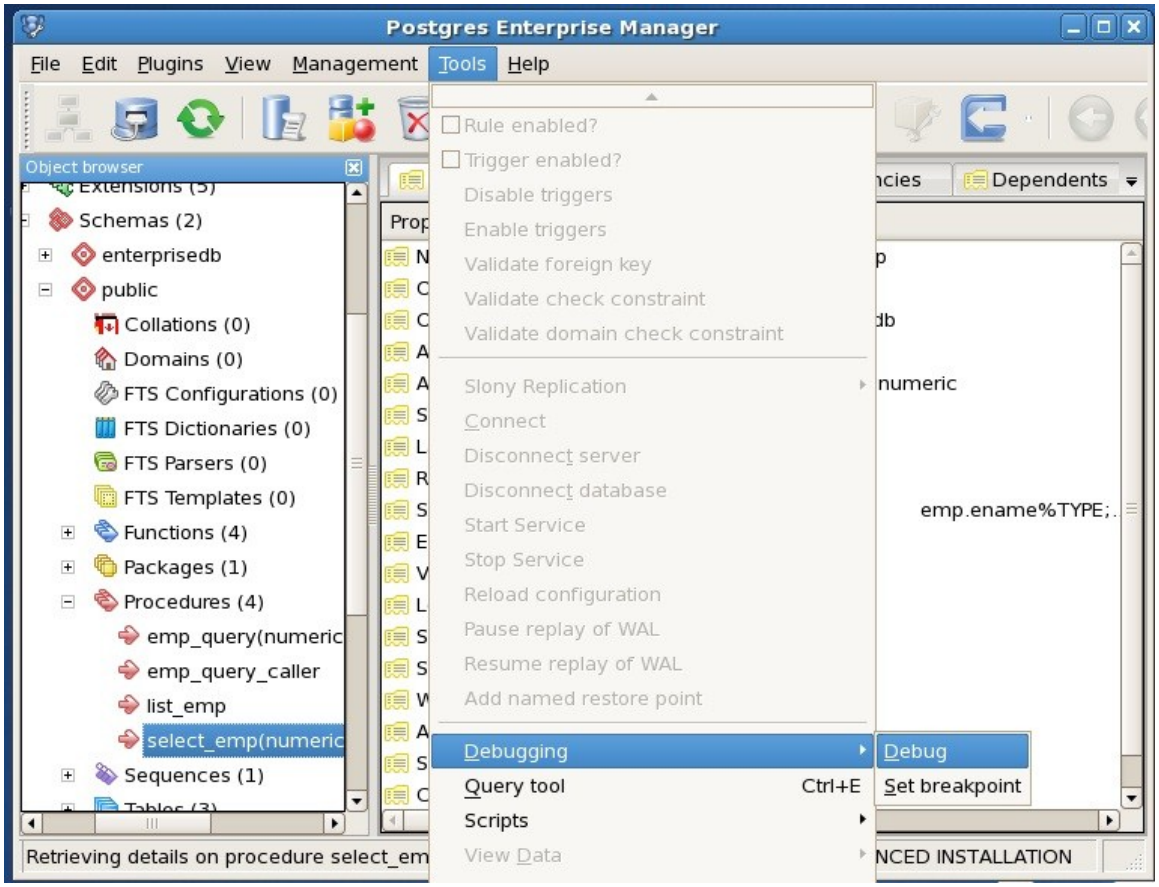

Figure 2 - Starting the Debugger from the Tools menu

- Alternatively, in the Postgres Enterprise Manager Client Object Browser, click the secondary mouse button on the stored procedure or function you wish to debug. This action opens the object's context menu from which you choose Debugging, then Debug from the submenu as shown by the following:

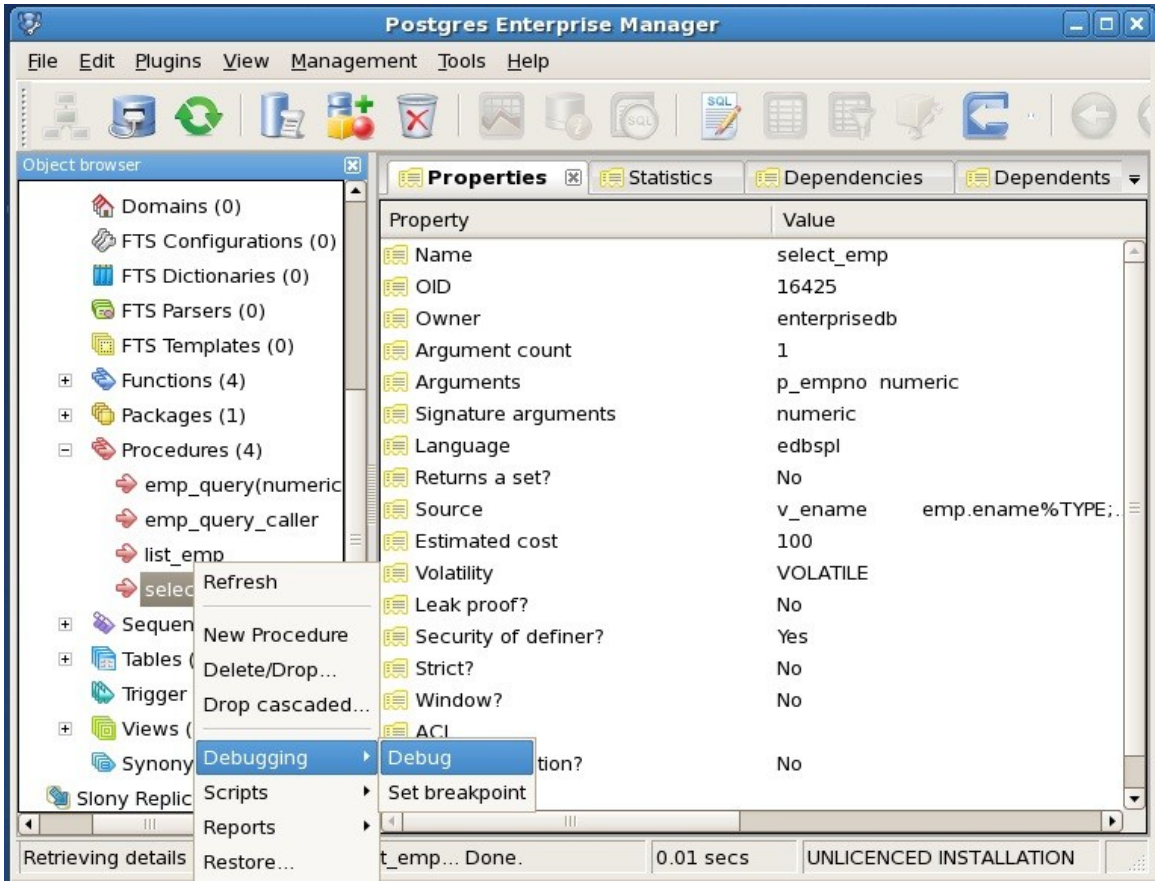

Figure 3 - Starting the Debugger from the object's context menu

Note: Triggers cannot be debugged using standalone debugging. Triggers must be debugged using in-context debugging. See Section 4.2.5.3 for information on setting a global breakpoint for in-context debugging.

Note: 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.

4.2.3 Parameter Grid Window

After starting the Debugger for standalone debugging, if the program has any `IN` or `IN OUT` parameters, the Parameter Grid window opens in which you can pass actual parameter values to the program being debugged. Only `IN` and `IN OUT` parameter types are shown in the Parameter Grid window.

If the program has no `IN` or `IN OUT` parameters, the Main Debugger window immediately opens instead of the Parameter Grid window. The Main Debugger window is described in Section 4.2.4.

Note: The Parameter Grid window does not appear during in-context debugging since it is the responsibility of the application calling the program to be debugged to supply any required input parameter values.

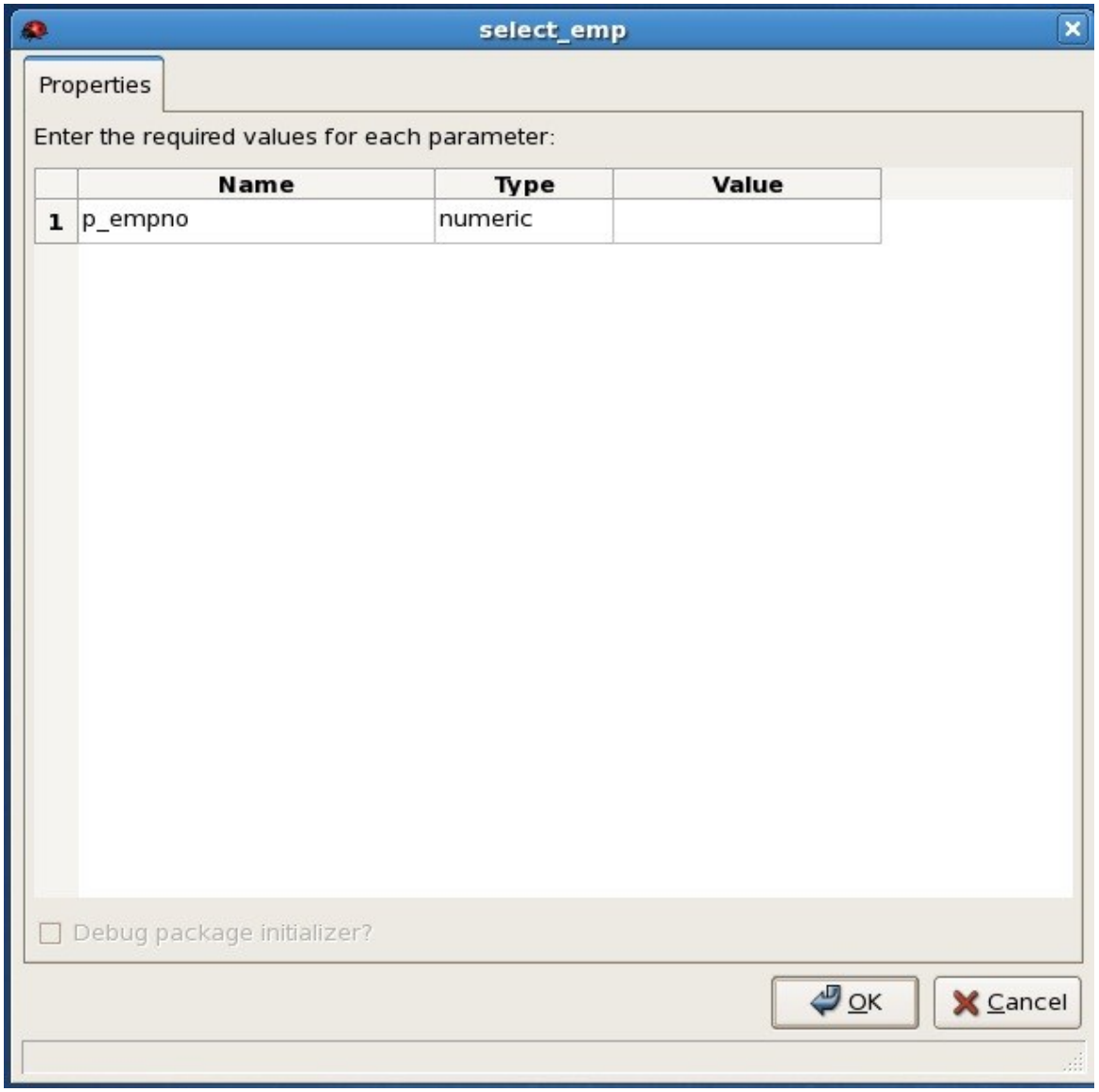

Figure 4 - Parameter Grid window

The Parameter Grid window displays a table with a list of formal parameter names, their data types, and values to be passed to the program.

For each parameter, type the value to be passed in the Value text box. If nothing is entered in the text box, the corresponding parameter is set to null. Press the Enter key to select the next parameter in the list for data entry, or click on any Value text box to select its parameter for data entry.

Note: If you are debugging a procedure or function that is a member of a package that has an initialization section, you can check the Debug Package Initializer check box in which case the Debugger steps into the package initialization section allowing you to debug the initialization section code before debugging the procedure or function. If you do not 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 you have entered the desired parameter values, click the OK button to start the debugging process.

Note: If you click the Cancel button, the Debugger terminates and control returns back to Postgres Enterprise Manager Client.

After you have completed a full debugging cycle by stepping through the program code as described in Section 4.2.5, the Parameter Grid window re-opens. You can enter new parameter values and click the OK button to repeat the debugging cycle, or you can end the debugging session by clicking the Cancel button and then choosing Exit from the File menu as shown in Section 4.2.5.4.

4.2.4 Main Debugger Window

The Main Debugger window has three frames, which are the Program Body frame, the Call Stack frame, and the Variable View frame. A menu bar provides access to the available operations and a tool bar displays icons for quick access to available debugging operations. In addition, status and error information are displayed in an information bar at the bottom of the Main Debugger window.

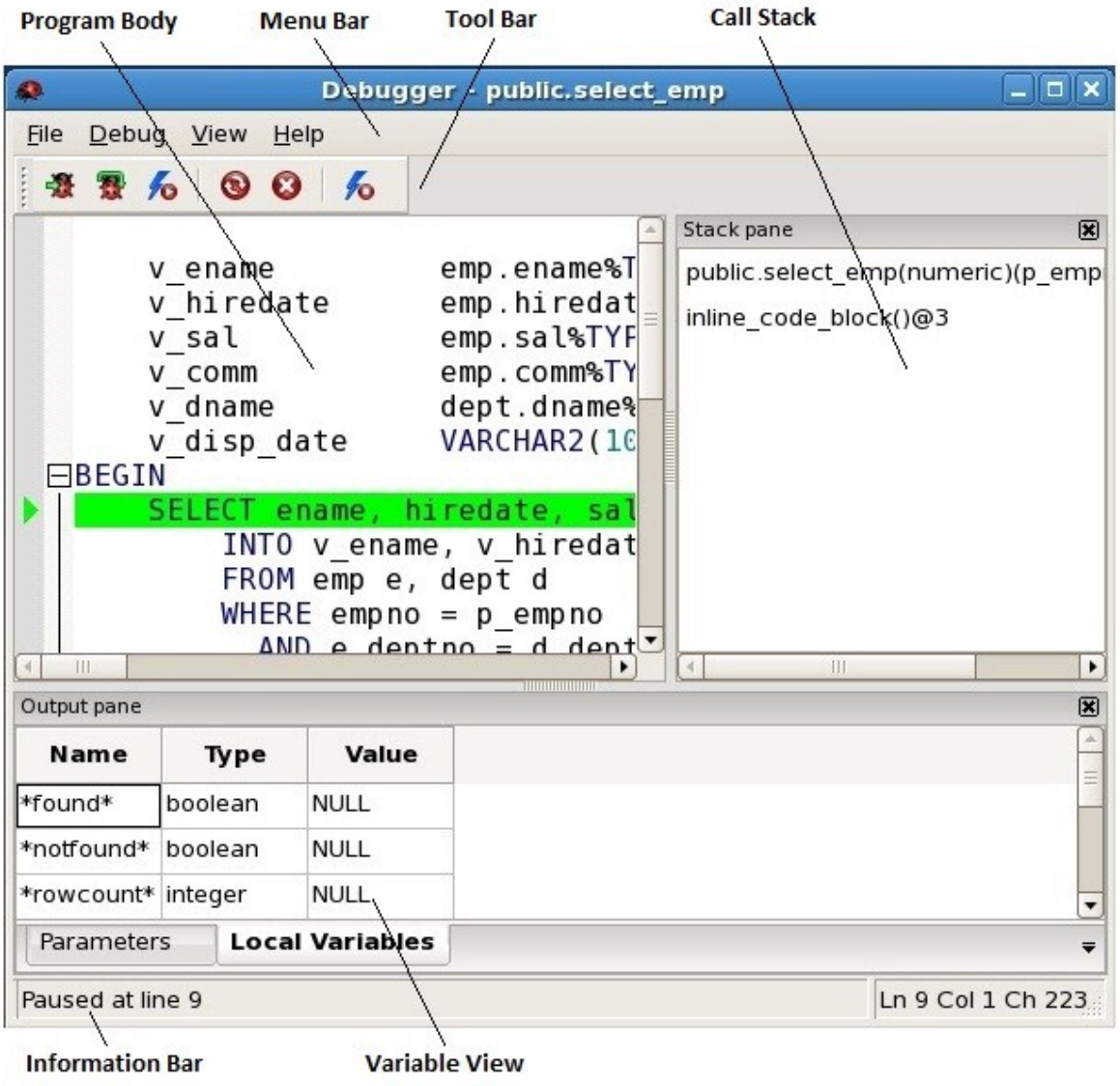

Figure 5 - Main Debugger window

4.2.4.1 Program Body Frame

The Program Body frame displays the source code of the program being debugged.

A green bar highlights the background of the line of code that will be executed next.

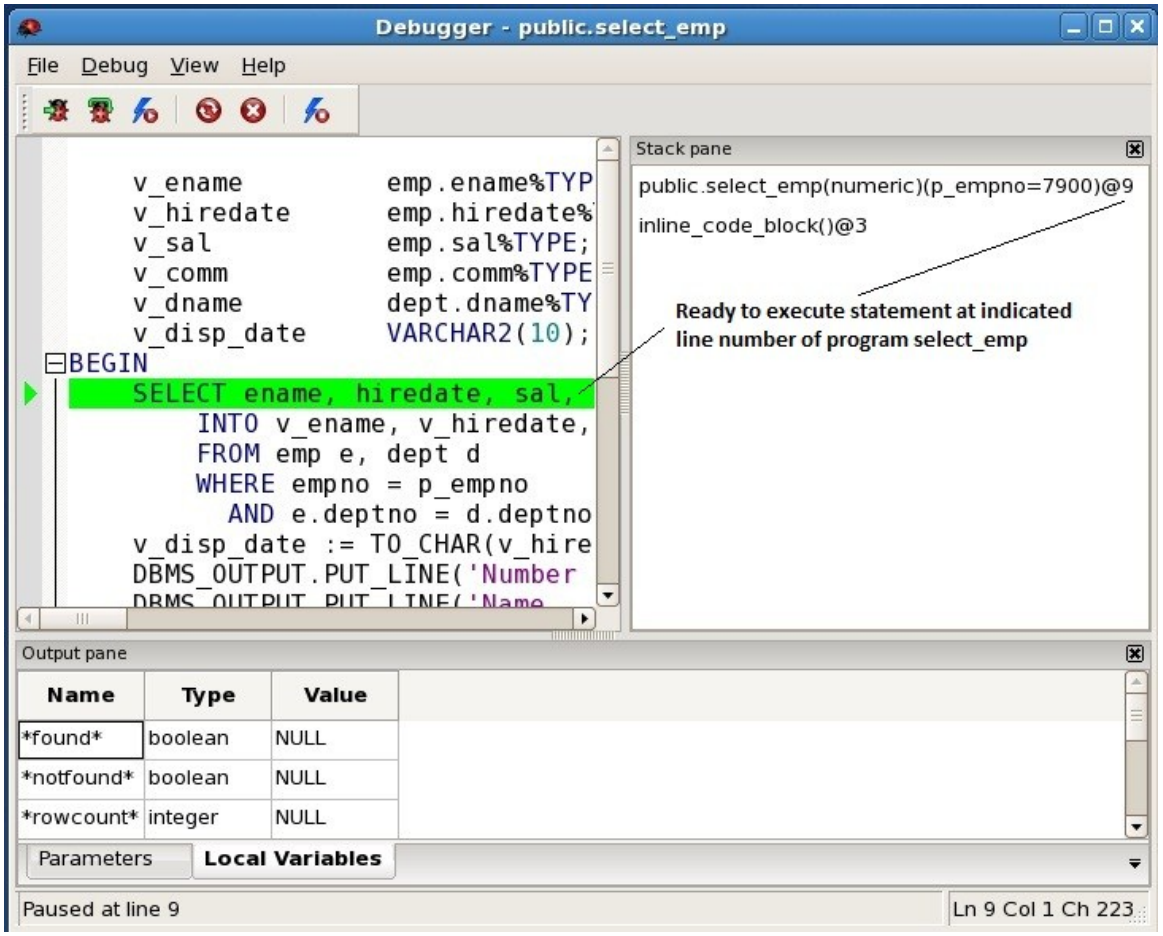

Figure 6 - Program Body frame

The `SELECT` SQL command is the next line of code waiting to be executed.

4.2.4.2 Call Stack Frame

While debugging, you can see a list of programs that are currently loaded. Each time a program is called, the name of the called program is added to the top of the Call Stack frame. When the program ends, its name is removed from the Call Stack frame.

Each time your program performs a program call, information about the call is generated in the Call Stack frame. That information includes the location of the call in your program, the arguments of the call, and the name of the program being called. Viewing the call stack can help you trace the course of execution through a series of nested programs.

The following example shows program `emp_query_caller` about to call subprogram `emp_query`. Program `emp_query_caller` is currently at the top of the Call Stack frame.

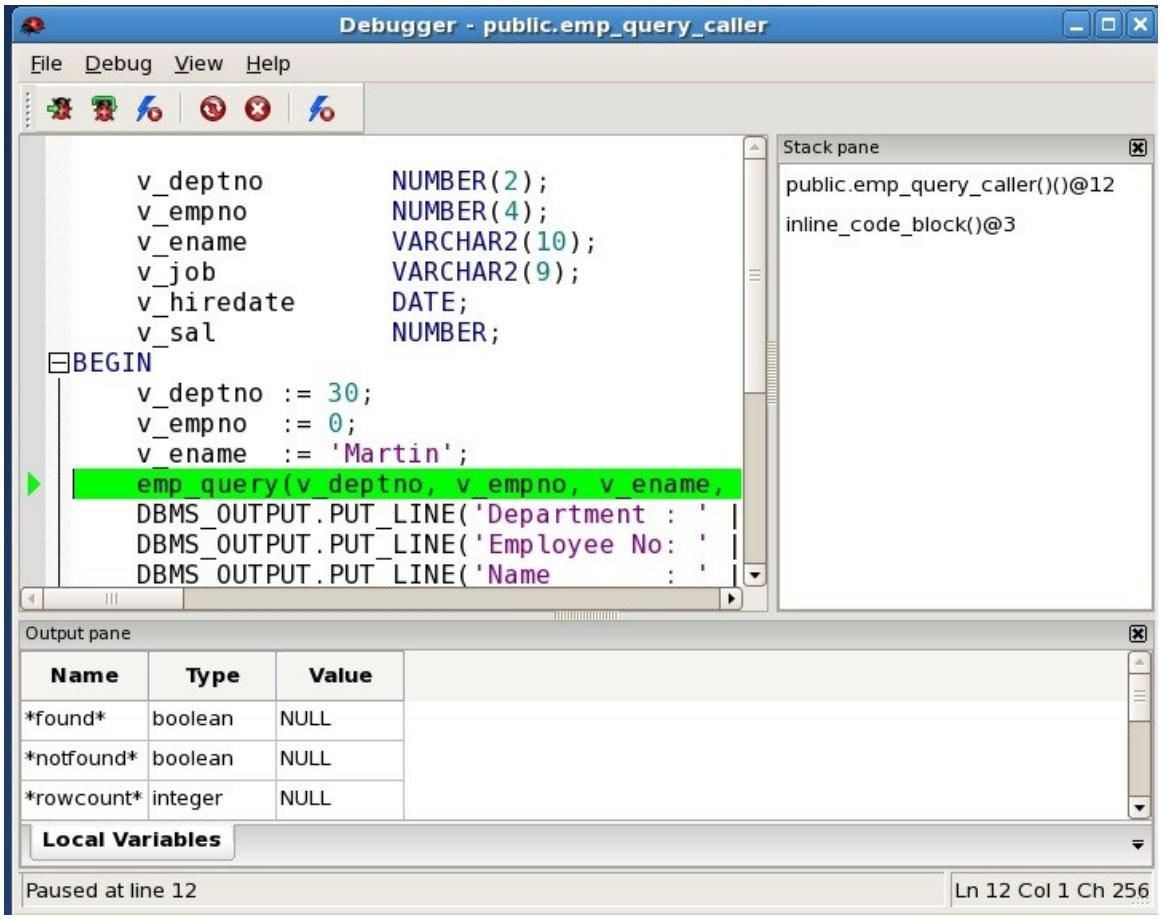

Figure 7 – Debugged program calling a subprogram

After the call to program `emp_query` is executed, `emp_query` now appears at the top of the Call Stack frame as its code is now appears in the Program Body frame as shown by the following:

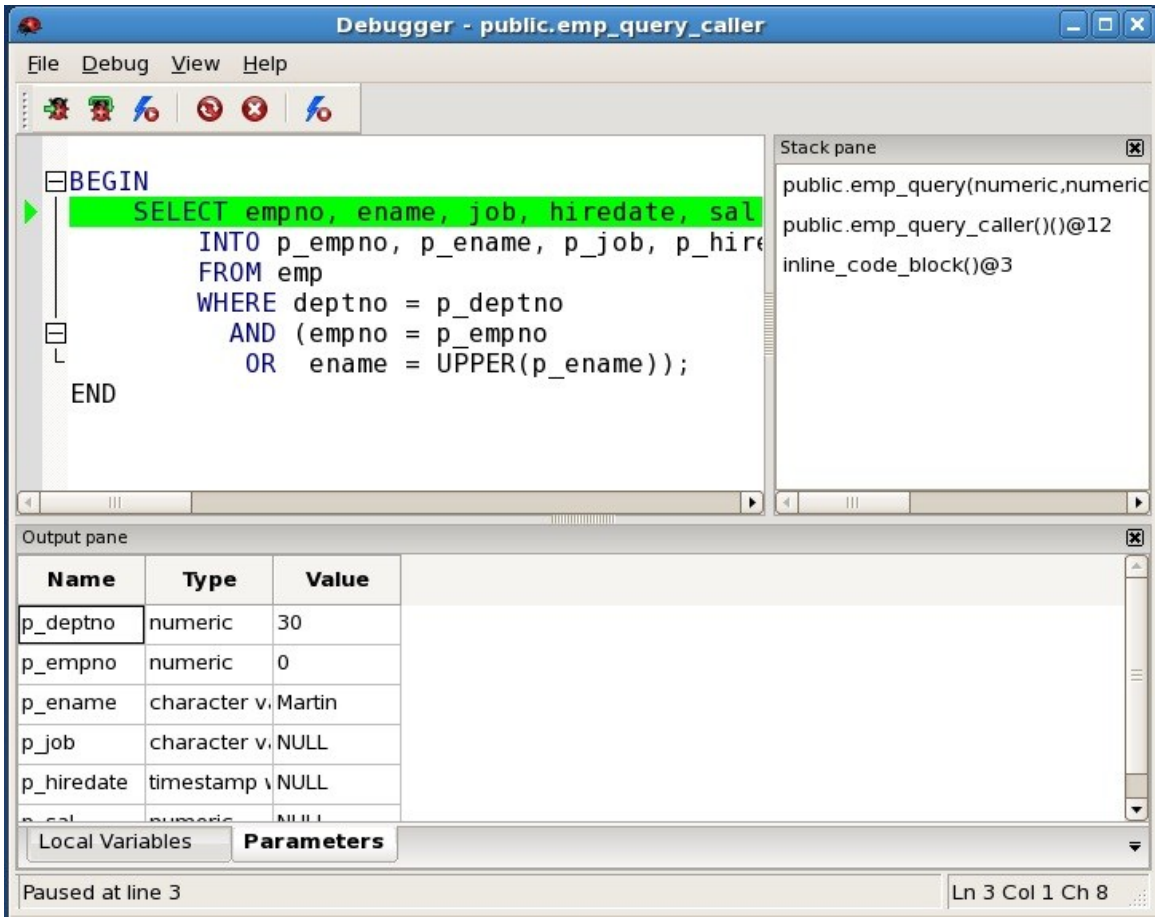

Figure 8 - Debugging the called subprogram

Upon completion of execution of the subprogram, control returns to the calling program. The top of the Call Stack frame now displays the calling program, `emp_query_caller` as shown by the following:

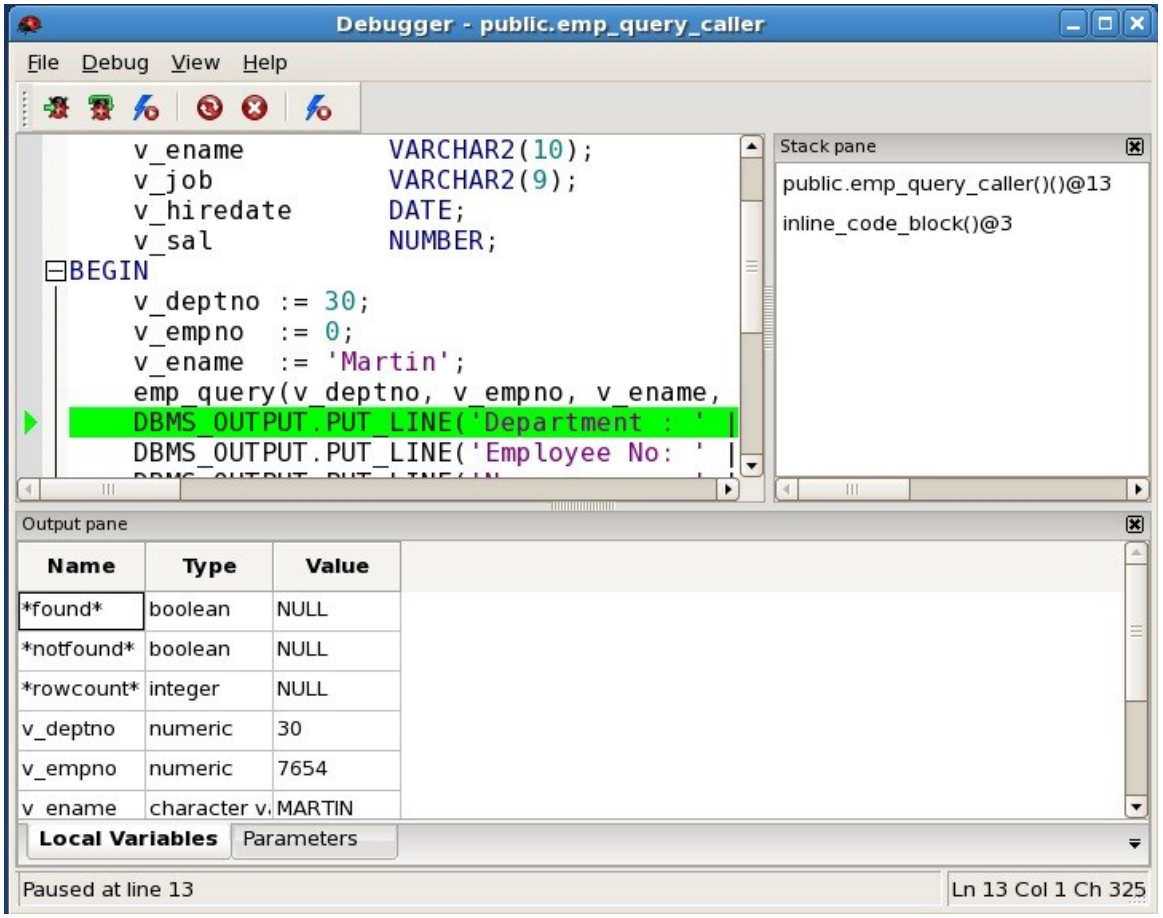

Figure 9 – Control returns from debugged subprogram

You can also use the call stack to navigate to a specific program. When you navigate to another program, you can set breakpoints, see variable values, and so on.

However, using the call stack to navigate to another program does not change the currently executing line. Even if you set a breakpoint or variable value in another program, execution resumes from the line where the program was halted.

4.2.4.3 Variable View Frame

While debugging, you can view and change the values of variables and parameters. The parameters and local variables are accessible from their respective tabs in the Variable View frame.

The following shows the input parameter for the `select_emp` program in the Parameters tab of the Variable View frame:

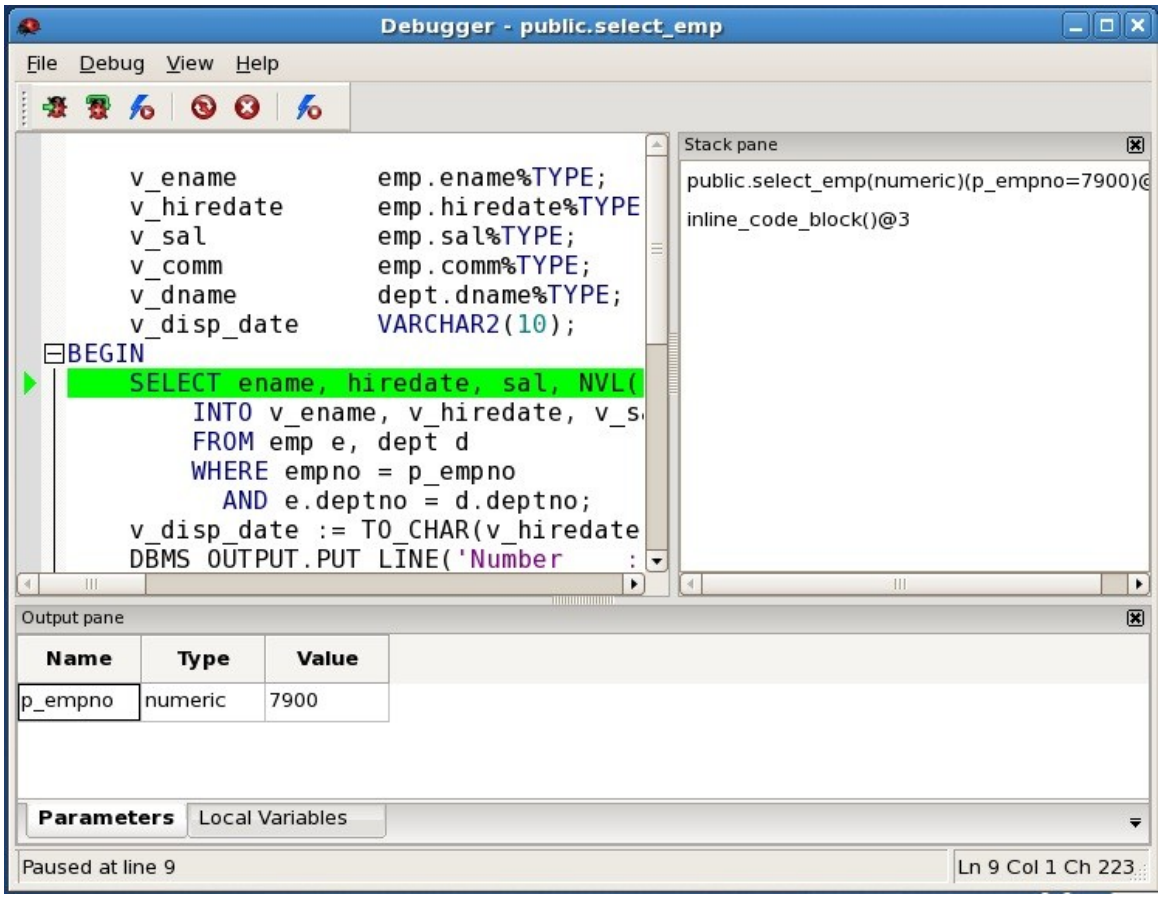

Figure 10 – Parameters tab of the Variable View frame

The following shows the local variables for the `select_emp` program in the Local Variables tab of the Variable View frame:

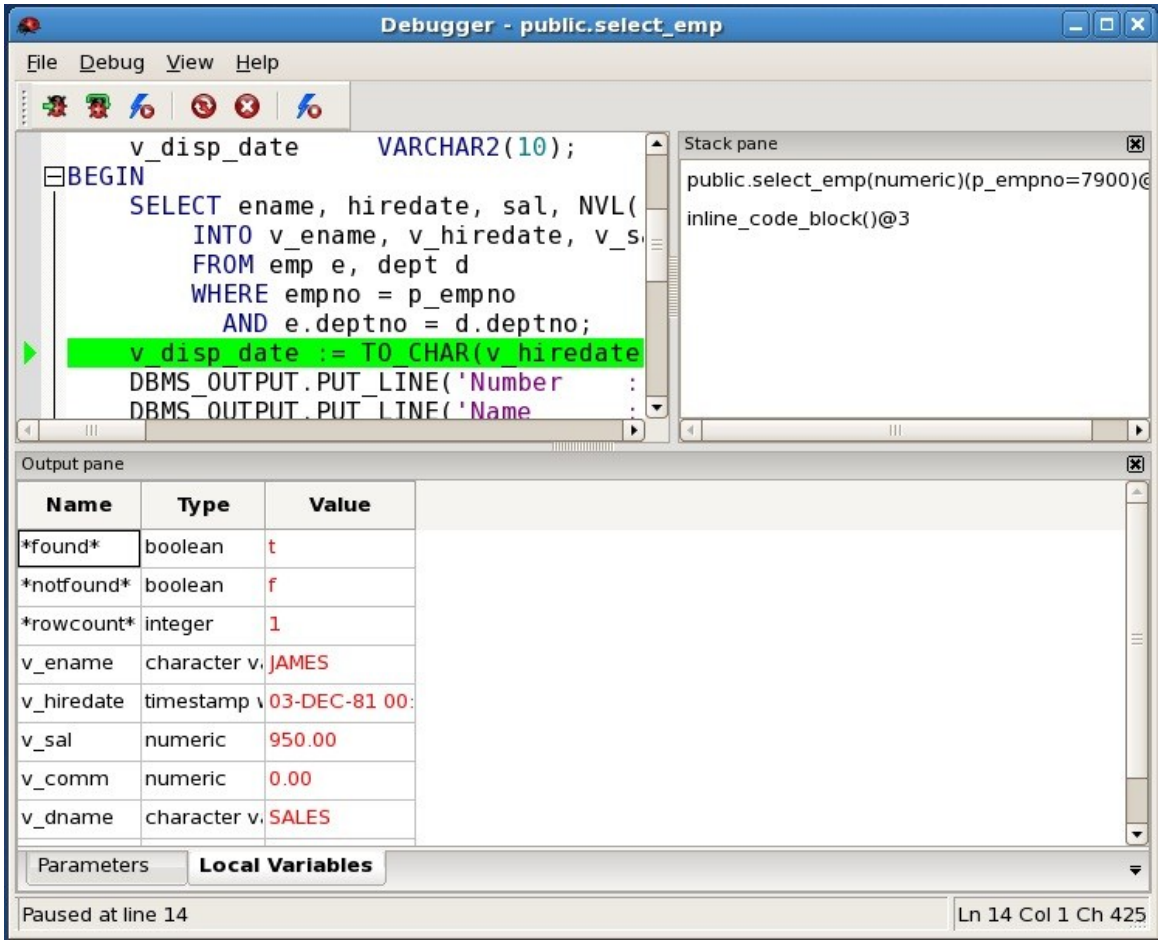

Figure 11 - Local Variables tab of the Variable View frame

4.2.4.4 Information Bar

The information bar displays a message when the Debugger pauses or when any runtime error messages are encountered during the debugging session.

Figure 12 - Information bar

4.2.5 Debugging a Program

You can perform the following operations to debug a program:

- Step through and execute code one line at a time
- Execute a series of code lines pausing at selected breakpoints
- View and change local variable values

These are discussed in the following sections.

4.2.5.1 Stepping Through the Code

The following describes the three operations you can use to run the code of the program being debugged:

- **Step Into.** Executes the line of code highlighted by the green bar in the Program Body frame and then pauses execution at the next line of code which is then highlighted by the green bar. If the executed code line is a call to a subprogram, the called subprogram is brought into the Program Body frame, and the first executable line of code of the subprogram is highlighted as the Debugger waits for you to perform an operation on the subprogram. To perform the step into operation, click the Step Into icon on the tool bar or press Function key F11 (Ctrl+F11 on Linux).
- **Step Over.** Same as the step into operation except if the executed line of code is a call to a subprogram containing no breakpoints, the called subprogram is executed in its entirety without bringing it into the Program Body frame. (If the called subprogram or any subprograms it may call contain breakpoints, the subprogram

with the breakpoint is brought into the Program Body frame and execution pauses at the line of code with the breakpoint.) To perform the step over operation, click the Step Over icon on the tool bar or press Function key F10 (Ctrl+F10 on Linux).

- **Continue.** Executes the line of code highlighted by the green bar and continues execution of successive code lines until either a breakpoint is encountered or the last line of the program has been executed. To perform the continue operation, click the Continue icon on the tool bar or press Function key F5 (Ctrl+F5 on Linux).

The following shows the locations of the Step Into, Step Over, and Continue icons on the tool bar:

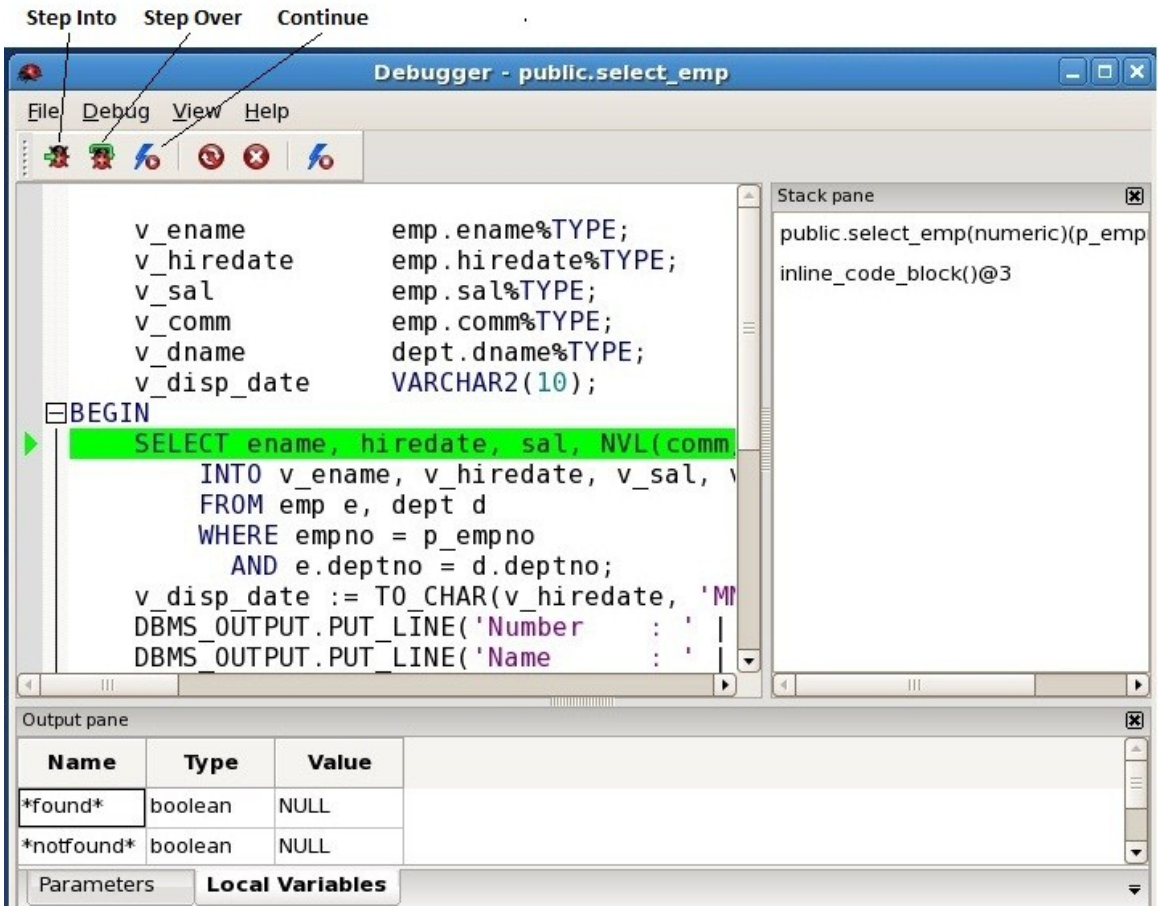

Figure 13 - Step Into, Step Over, and Continue icons

The debugging operations are also accessible through the Debug menu as shown by the following:

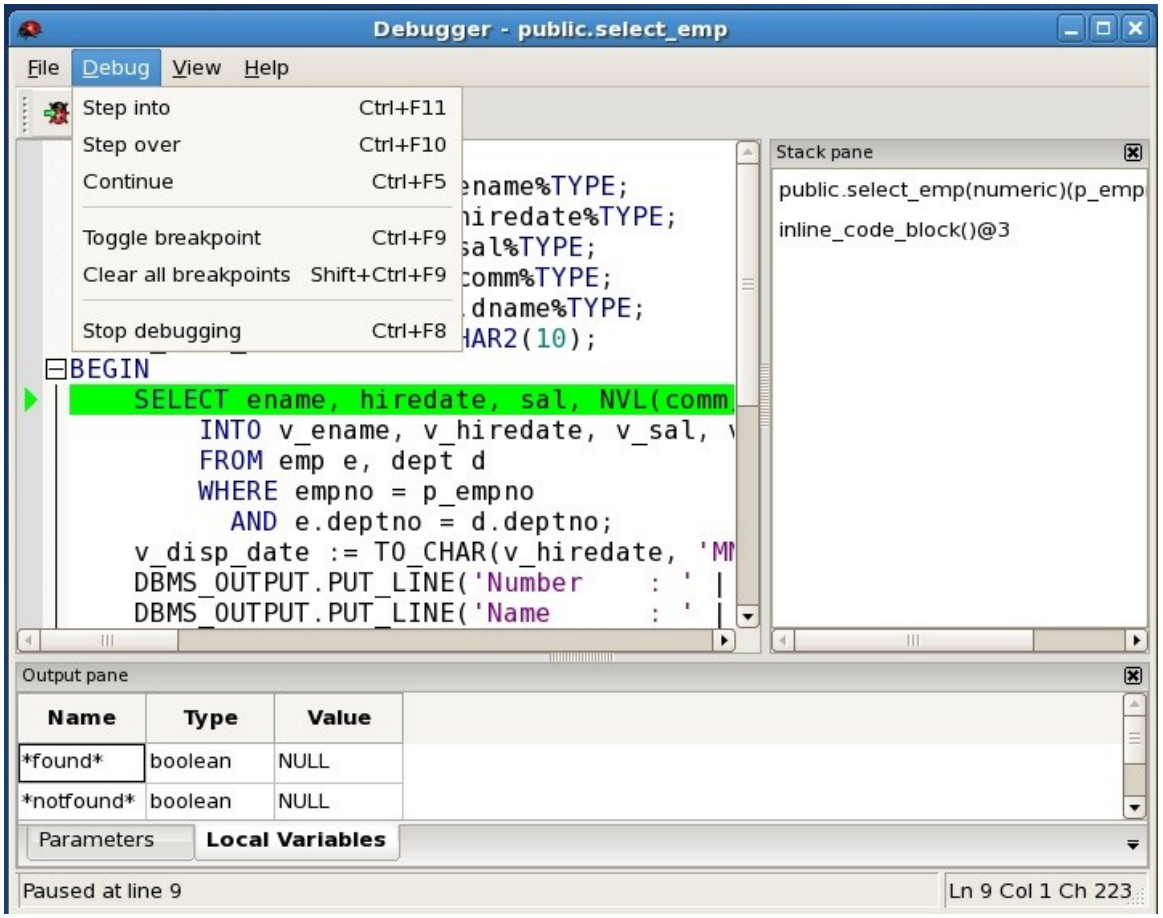

Figure 14 - Debug menu options

4.2.5.2 Using Breakpoints

As the Debugger executes through successive lines of code, it pauses whenever a point in the program is reached where a breakpoint has been set. When the Debugger pauses, you can observe or change local variables, or navigate to another program in the call stack to observe its 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.** Can be set at any executable line of code within the program being debugged. When the Debugger is executing through successive lines of code, the Debugger pauses execution when it reaches a line where a local breakpoint has been set.
- **Global Breakpoint.** Set once for the program being debugged. 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 the program to be debugged is invoked by an application running in another session. A global breakpoint can only be set by a superuser.

The remainder of this section discusses how to use local breakpoints. For setting a global breakpoint for in-context debugging see Section 4.2.5.3.

A local breakpoint can be set in any of the following ways:

- In the Program Body frame, click the primary mouse button in the shaded, left-hand margin next to the line of code where you want a local breakpoint set. A red dot appears in the margin indicating a breakpoint has been set at the line of code.

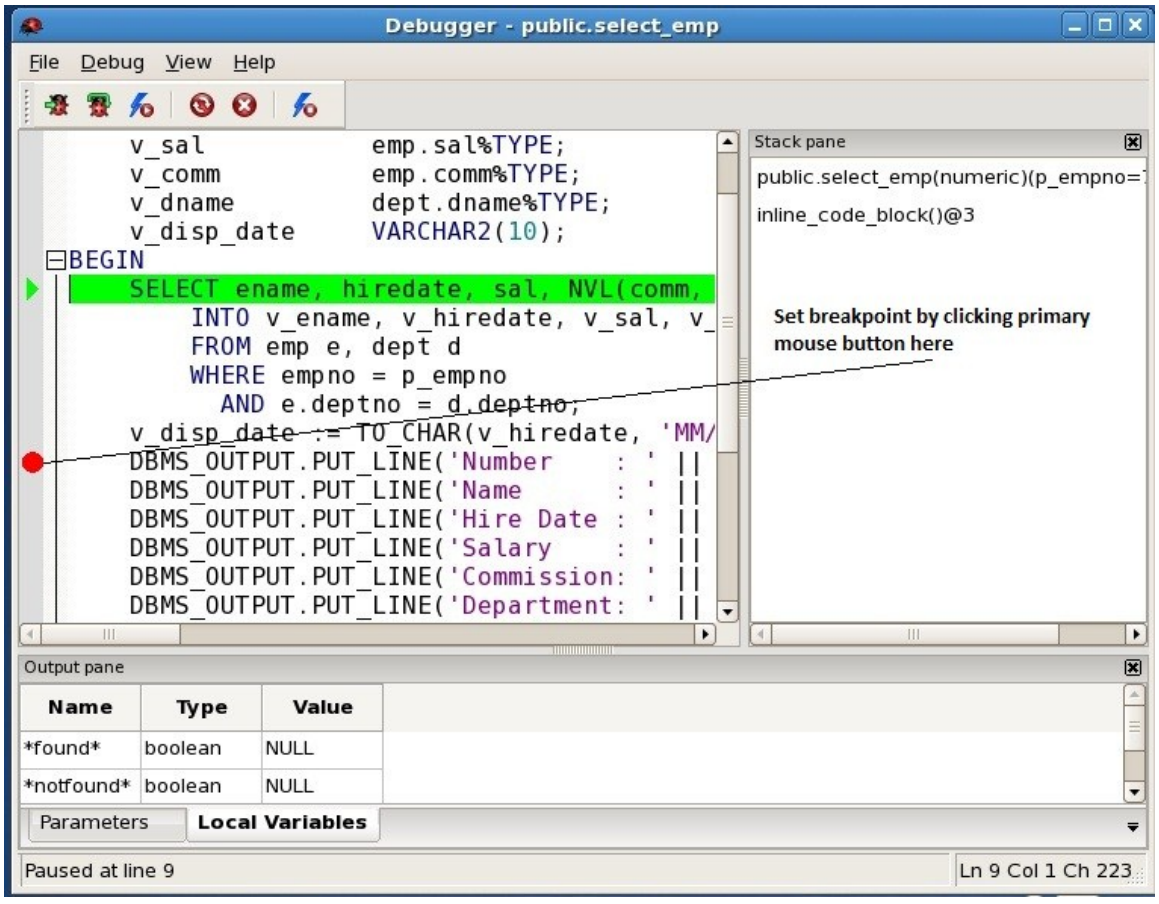

Figure 15 - Set breakpoint by clicking in left-hand margin

- Alternatively, in the Program Body frame, click the primary mouse button in the main, white code area so the cursor is located on the line of code where you want a local breakpoint set. Then perform any one of the following actions: 1) From the menu bar, select Debug, then choose Toggle Breakpoint. 2) On the tool bar, click the Toggle Breakpoint icon. 3) Press Function key F9 (Ctrl+F9 on Linux).

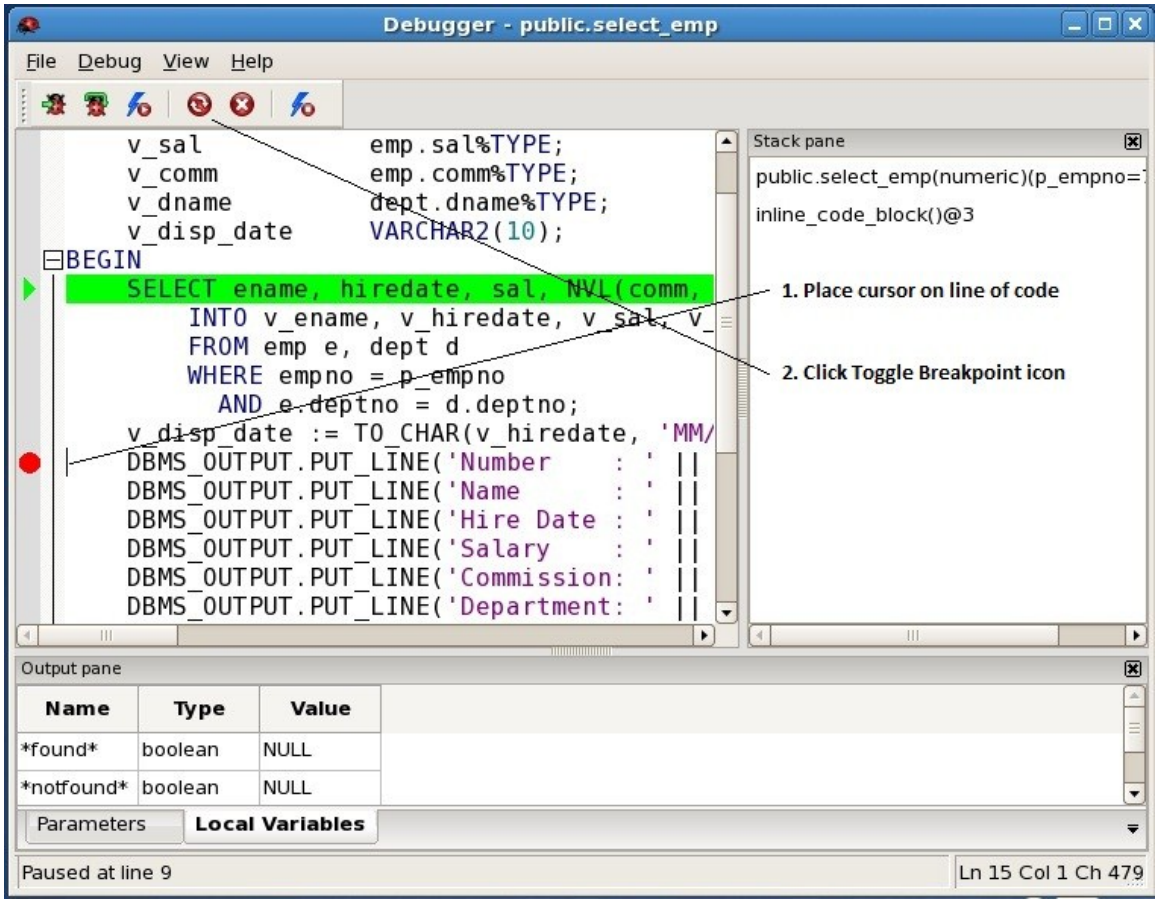

Figure 16 – Set breakpoint by selecting line of code then clicking Toggle Breakpoint icon

A red dot appears in the left-hand margin indicating a breakpoint has been set at the line of code.

You can set as many local breakpoints as desired.

The Debugger suspends execution when a breakpoint is encountered as shown by the green highlight bar on the line of code with a breakpoint.

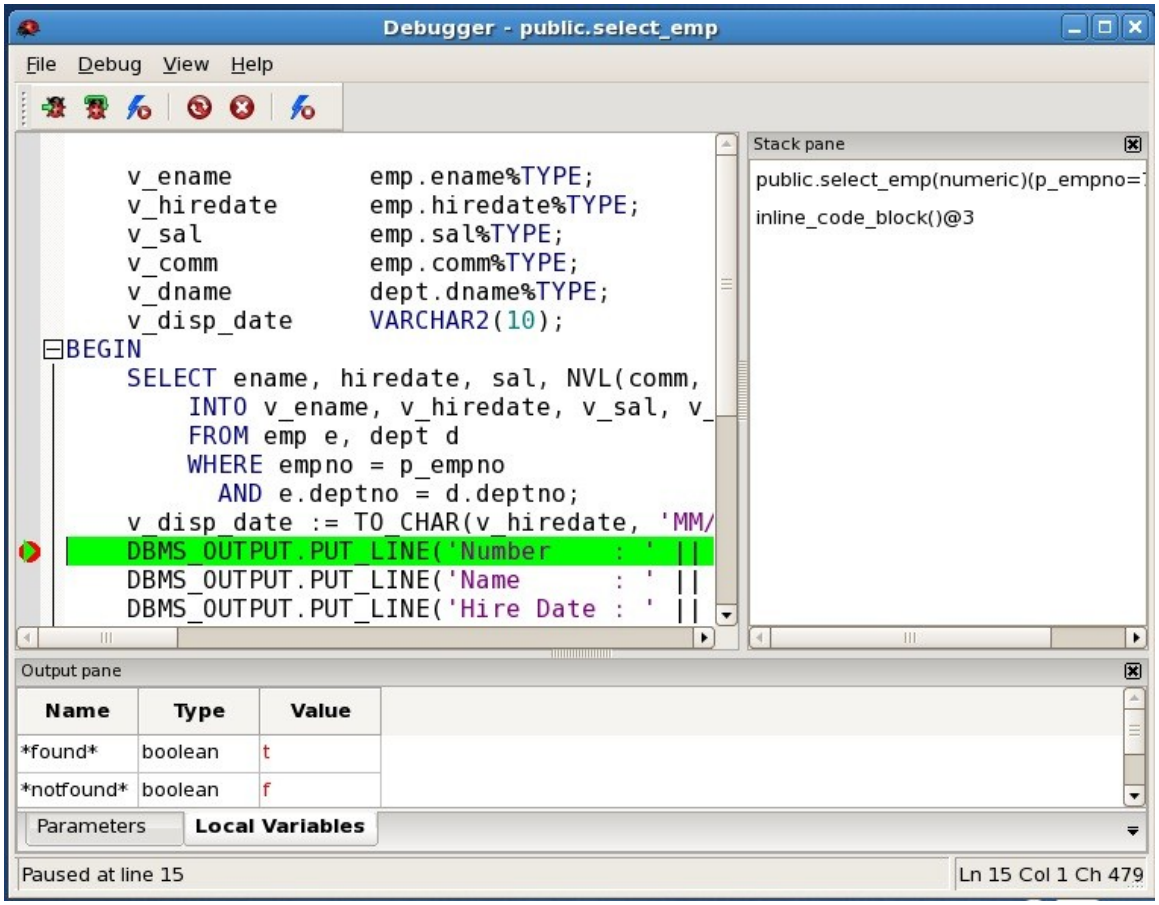

Figure 17 - Debugger paused at a breakpoint

Use any of the step into, step over, or continue operations previously discussed to execute the line of code.

Local breakpoints remain in effect for the duration of a debugging session until they are removed as discussed in the following section.

Removing a Local Breakpoint

Removing a local breakpoint is done in a manner similar to setting a breakpoint except you perform the operation on a line of code on which a breakpoint has already been set as indicated by a red dot. Use either of the following two methods:

- In the Program Body frame, click the primary mouse button in the shaded, left-hand margin on the red dot where a local breakpoint has been set. The red dot disappears indicating that the breakpoint has been removed.
- In the Program Body frame, click the primary mouse button in the main, white code area so the cursor is located on the line of code where a local breakpoint has been set. Then perform any one of the following actions: 1) From the menu bar, select Debug, then choose Toggle Breakpoint. 2) On the tool bar, click the Toggle

Breakpoint icon. 3) Press Function key F9 (Ctrl+F9 on Linux). The red dot disappears indicating that the breakpoint has been removed.

You can remove all breakpoints from the program that currently appears in the Program Body frame by performing any one of the following actions: 1) From the menu bar select Debug, then choose Clear All Breakpoints. 2) On the tool bar, click the Clear All Breakpoints icon. 3) Press Function key Ctrl+Shift+F9.

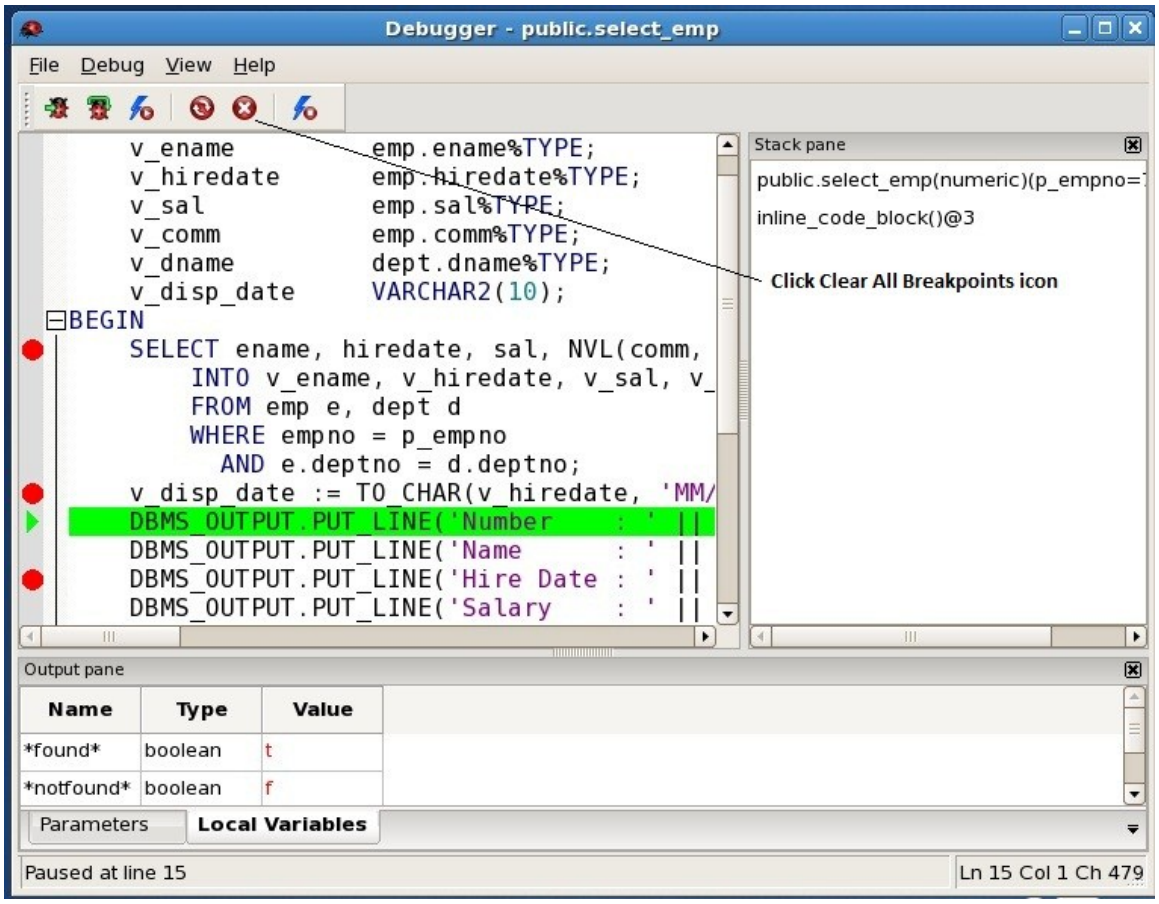

Figure 18 - Clear all breakpoints

Note: When you perform any of the preceding actions, only the breakpoints in the program that currently appears in the Program Body frame are removed. Breakpoints in called subprograms or breakpoints in programs that call the program currently appearing in the Program Body frame are not removed.

4.2.5.3 Setting a Global Breakpoint for In-Context Debugging

You can set a global breakpoint for in-context debugging in either of the following two ways.

- In the Postgres Enterprise Manager Client Object Browser, highlight the stored procedure, function, or trigger on which you wish to set a global breakpoint. From the main menu bar, select the Tools menu, choose Debugging from the menu, then Set Breakpoint from the submenu as shown by the following:

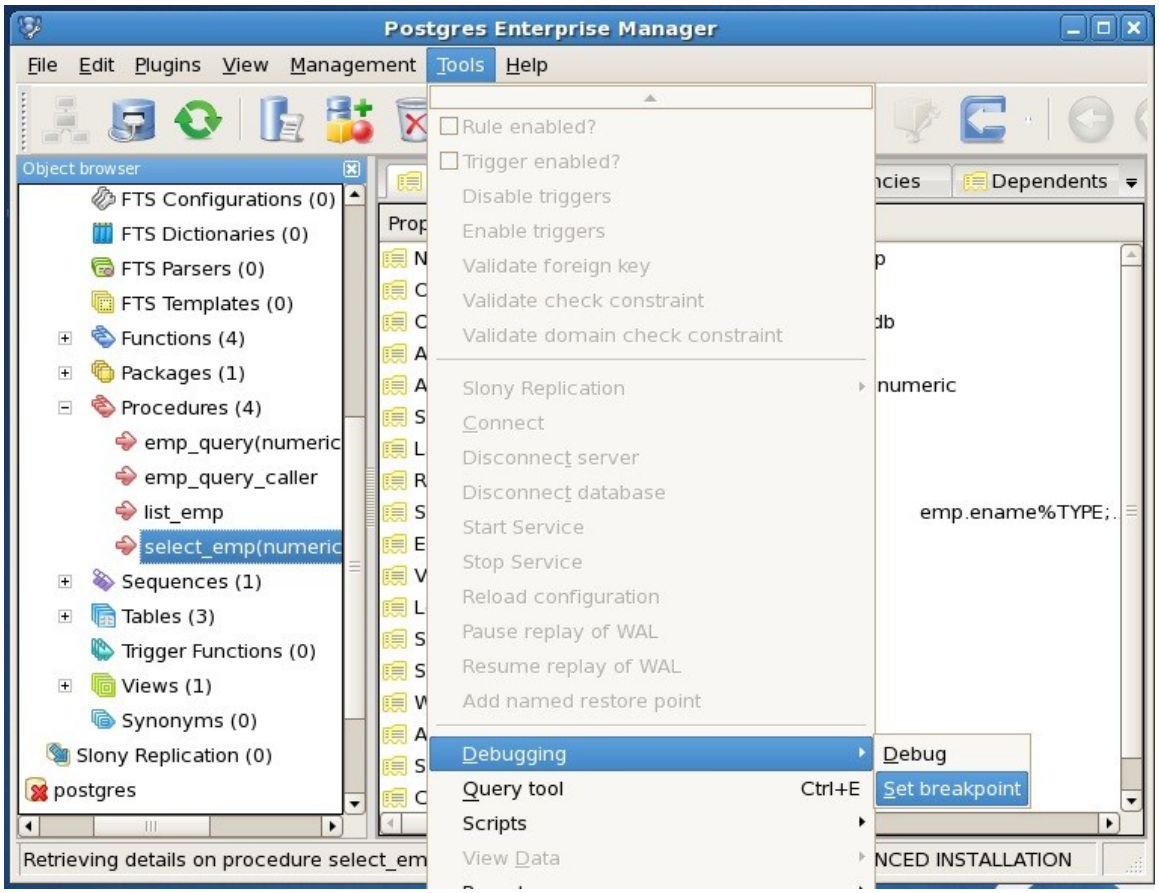

Figure 19 - Setting a global breakpoint from the Tools menu

- Alternatively, in the Postgres Enterprise Manager Client Object Browser, click the secondary mouse button on the stored procedure, function, or trigger on which you wish to set a global breakpoint. This action opens the object's context menu from which you choose Debugging, then Set Breakpoint from the submenu as shown by the following:

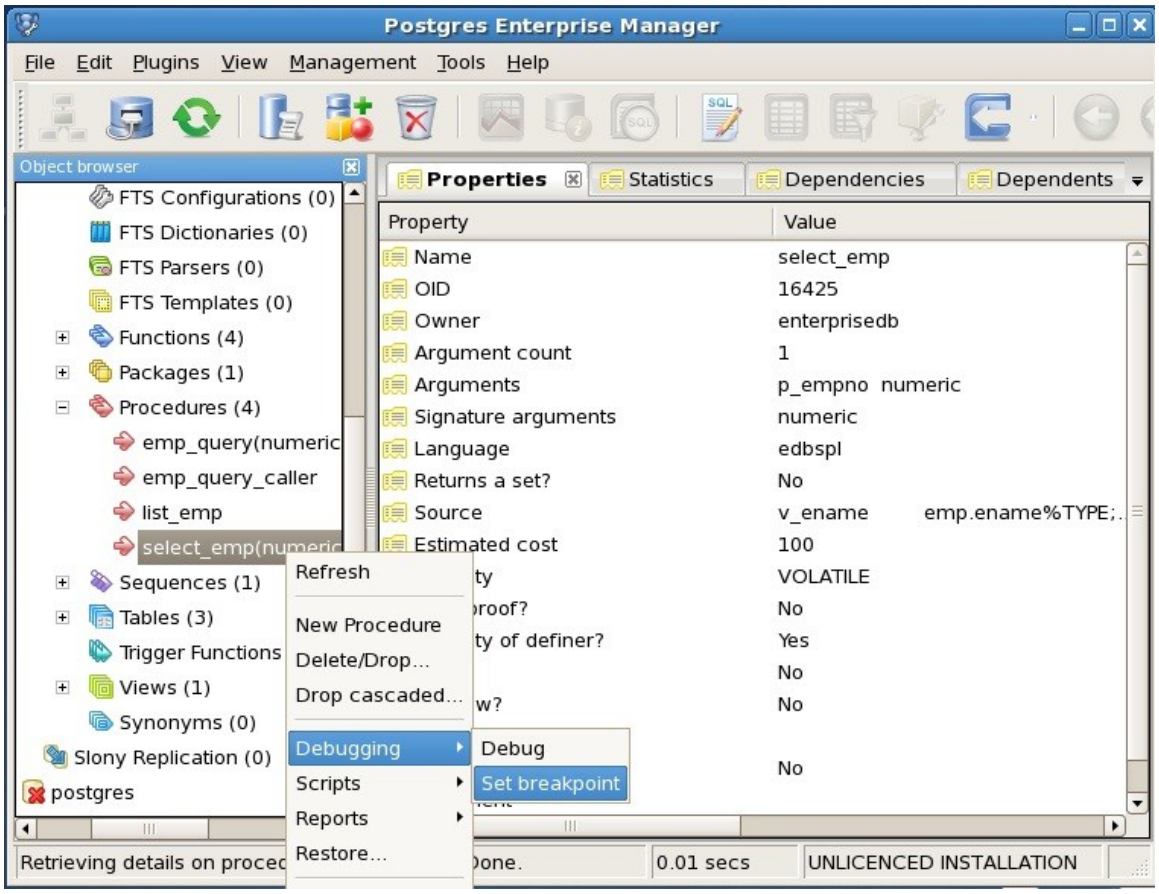

Figure 20 - Setting a global breakpoint from the object's context menu

Note: 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.

Note: 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 along with a status window indicating that the Debugger is waiting for some application to call the program to be debugged as shown by the following:

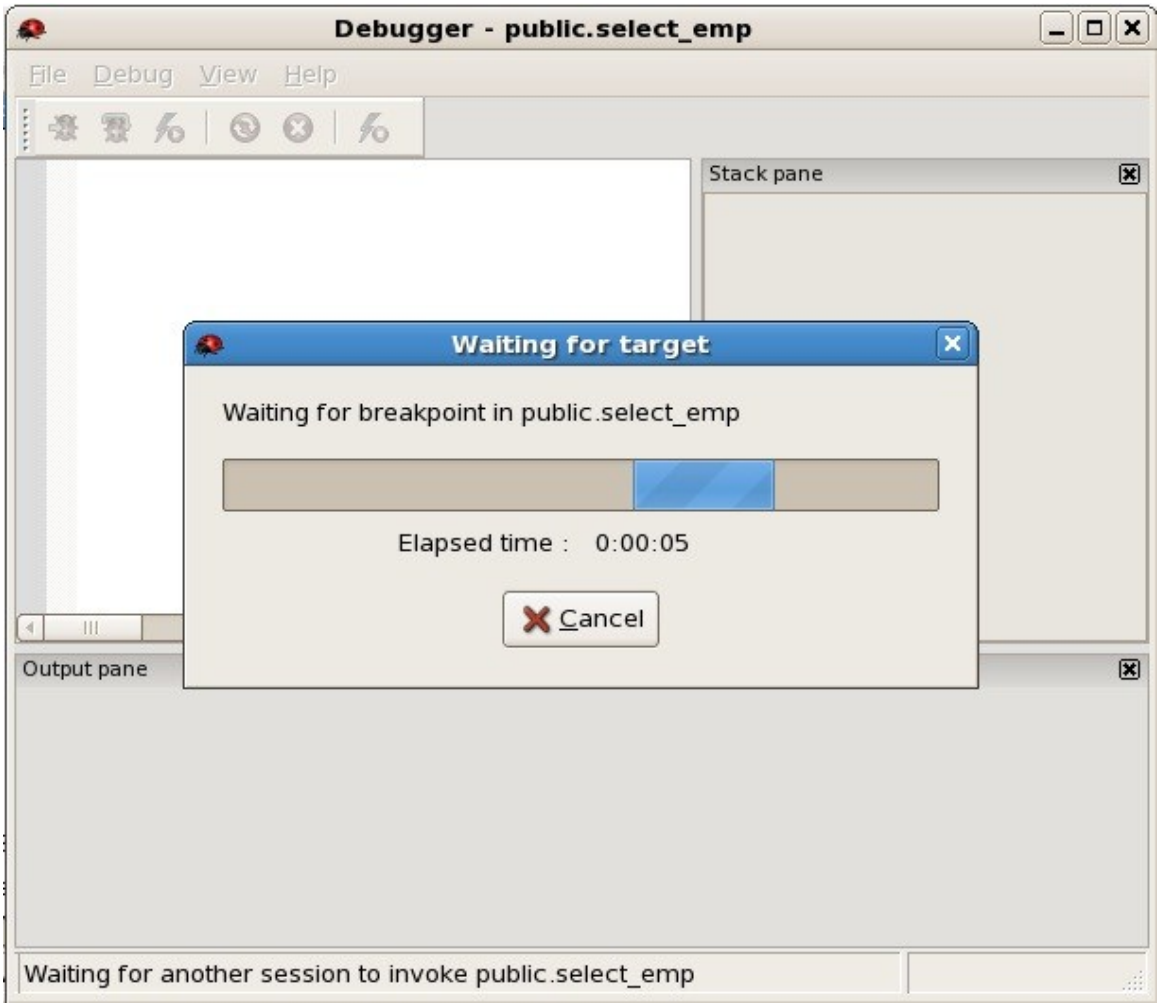

Figure 21 - Waiting for invocation of program to be debugged

In the following example, the EDB-PSQL utility program is the application used to invoke the `select_emp` procedure on which a global breakpoint has been set.

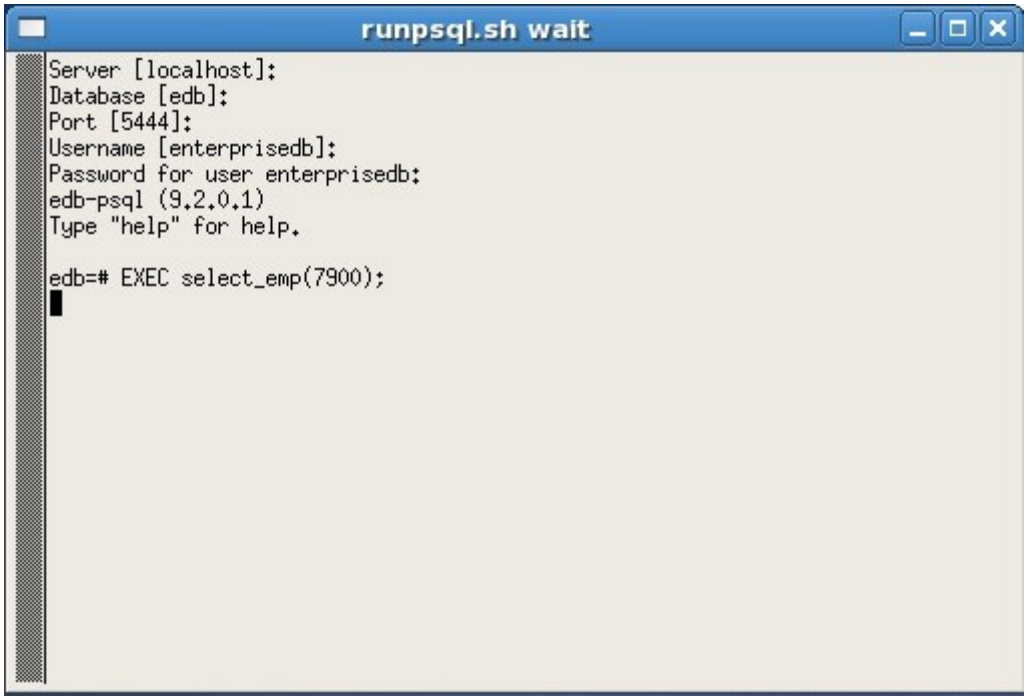

Figure 22 - Application invoking program with a global breakpoint

The `select_emp` procedure does not execute until you step through the program in the Debugger, which now appears as follows:

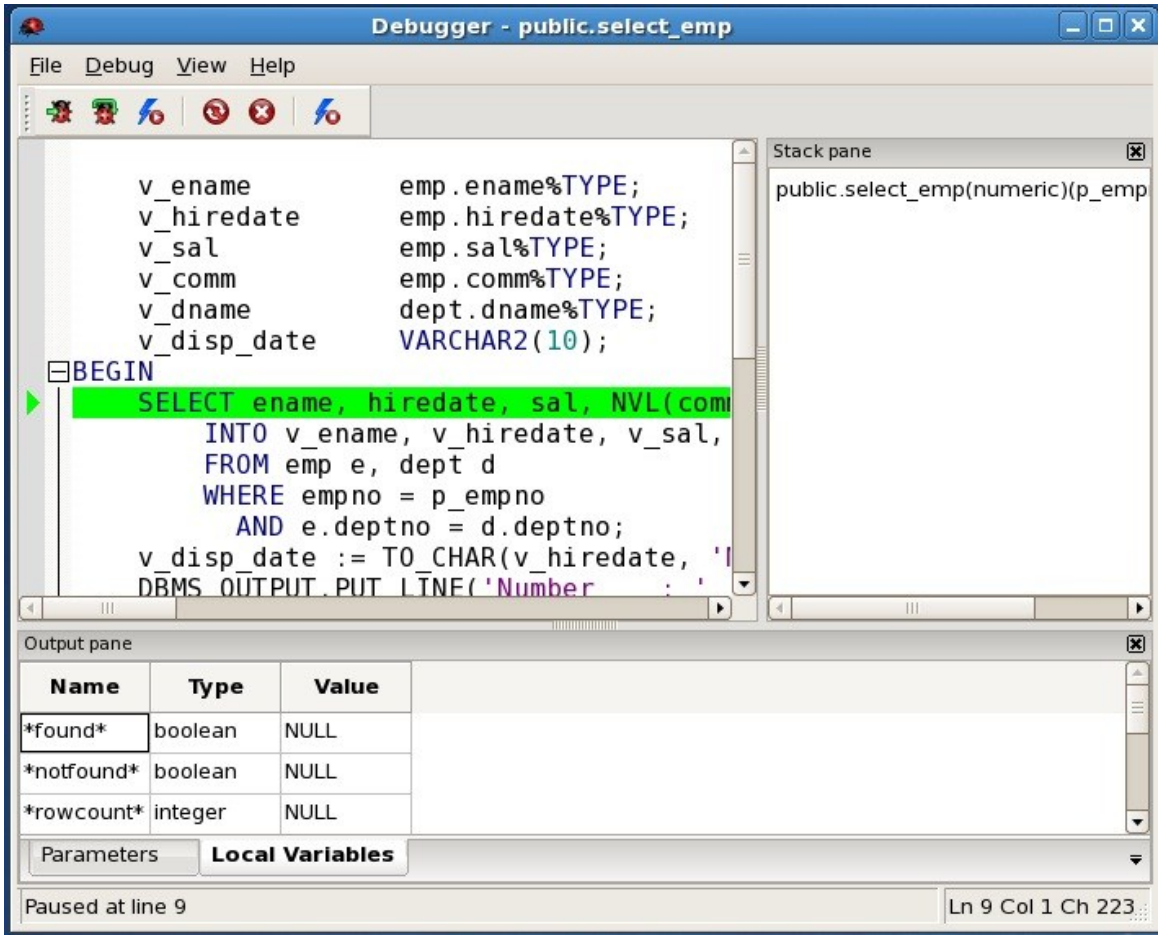

Figure 23 - Program on which a global breakpoint has been set

You can now debug the program using any of the previously discussed operations such as step into, step over, and continue. You can also set local breakpoints.

When you have completely stepped through execution of the program, the calling application (EDB-PSQL in this example) regains control as shown by the following:

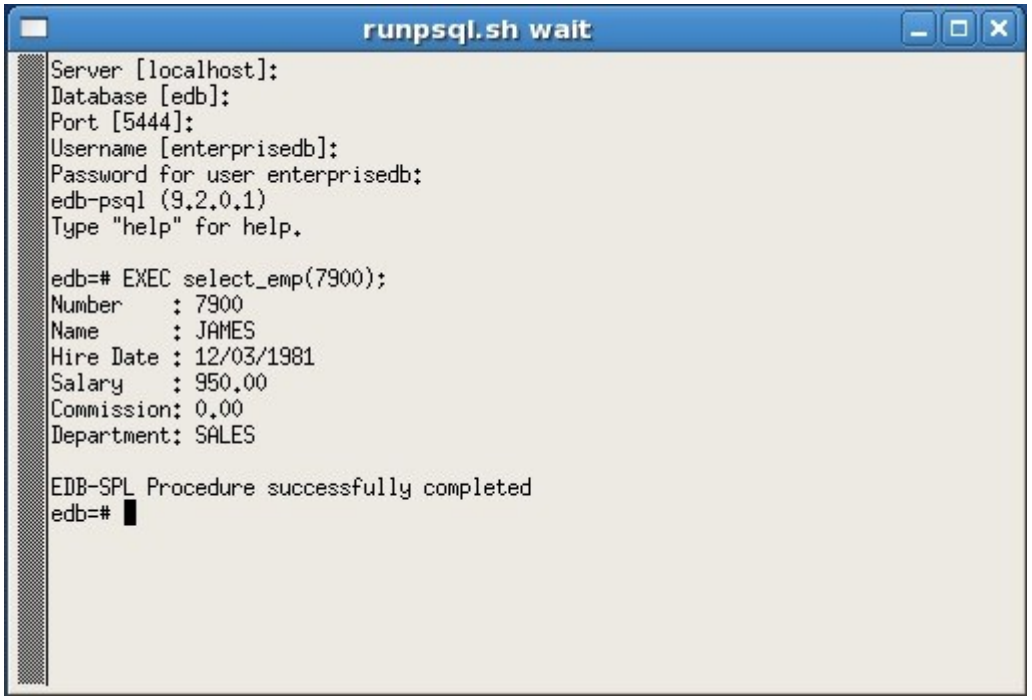

Figure 24 - Application after debugging

The `select_emp` procedure completes execution and its output is displayed.

At this point, you can end the Debugger session by choosing Exit from the File menu as shown in Section 4.2.5.4.

If you do not end the Debugger session, the next application that invokes the program encounters the global breakpoint and the debugging cycle begins again.

4.2.5.4 Exiting From the Debugger

You end the Debugger session by exiting from the Debugger. From the File menu, choose Exit or press Alt-F4 as shown by the following:

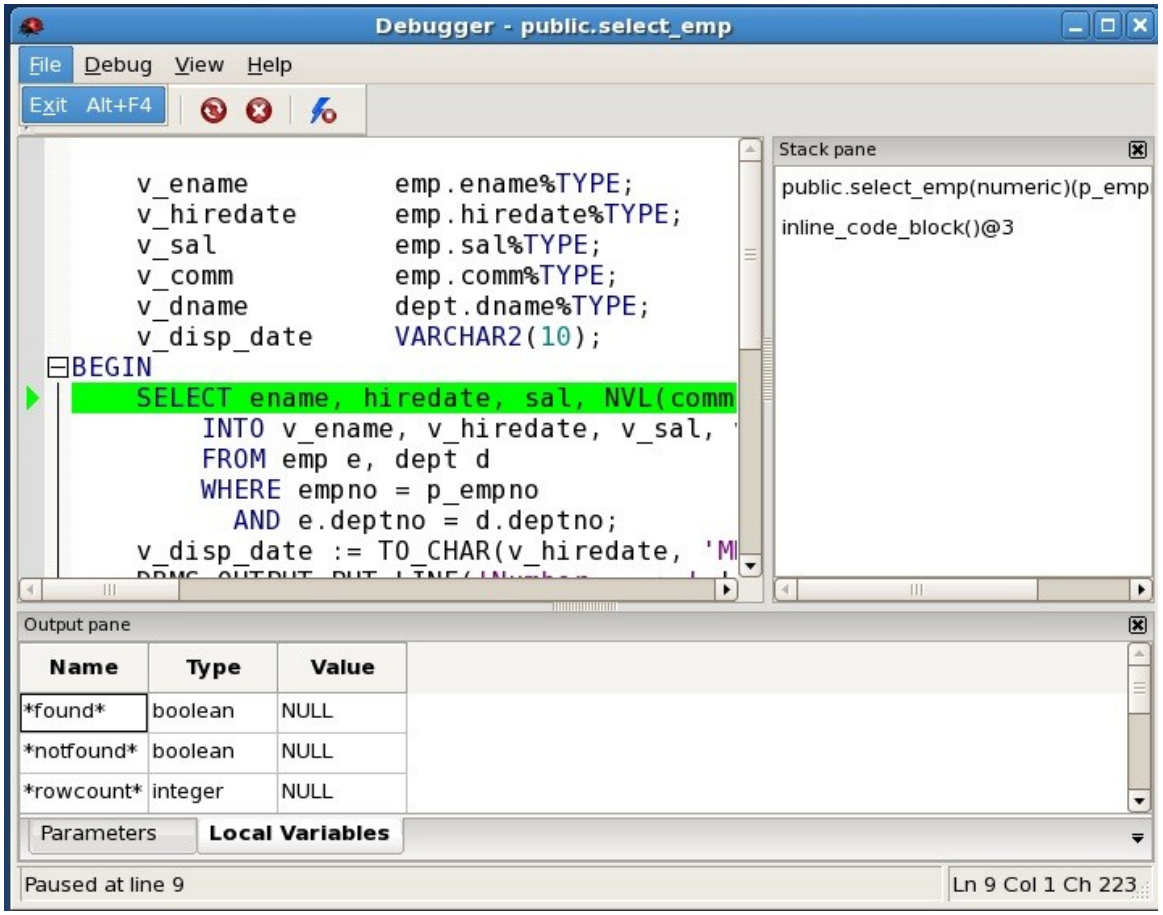

Figure 25 - Exiting from the Debugger