# Table of Contents

1 Introduction .......................................................................................................................... 5
   1.1 What’s New ...................................................................................................................... 6
   1.2 Typographical Conventions Used in this Guide ............................................................... 7
2 EDB*Plus ............................................................................................................................. 8
   2.1 Starting EDB*Plus ........................................................................................................... 8
   2.2 Command Summary ....................................................................................................... 11
      2.2.1 ACCEPT .................................................................................................................. 11
      2.2.2 APPEND ................................................................................................................ 11
      2.2.3 CHANGE ............................................................................................................... 12
      2.2.4 CLEAR .................................................................................................................. 12
      2.2.5 COLUMN .............................................................................................................. 13
      2.2.6 CONNECT .............................................................................................................. 16
      2.2.7 DEFINE ............................................................................................................... 17
      2.2.8 DEL ....................................................................................................................... 18
      2.2.9 DESCRIBE ............................................................................................................ 19
      2.2.10 DISCONNECT ...................................................................................................... 19
      2.2.11 EDIT .................................................................................................................... 20
      2.2.12 EXECUTE ............................................................................................................. 20
      2.2.13 EXIT ..................................................................................................................... 20
      2.2.14 GET ...................................................................................................................... 21
      2.2.15 HELP ................................................................................................................... 22
      2.2.16 HOST ................................................................................................................... 22
      2.2.17 INPUT .................................................................................................................. 22
      2.2.18 LIST ..................................................................................................................... 23
      2.2.19 PASSWORD ......................................................................................................... 24
      2.2.20 PAUSE .................................................................................................................. 24
      2.2.21 PRINT ................................................................................................................... 24
      2.2.22 PROMPT .............................................................................................................. 24
      2.2.23 QUIT ..................................................................................................................... 25
      2.2.24 REMARK .............................................................................................................. 25
      2.2.25 SAVE .................................................................................................................... 25
      2.2.26 SET ...................................................................................................................... 26
      2.2.27 SHOW ................................................................................................................... 31
      2.2.28 SPOOL .................................................................................................................. 31
      2.2.29 START ................................................................................................................... 32
      2.2.30 UNDEFINE .......................................................................................................... 32
      2.2.31 WHENEVER SQLERROR ...................................................................................... 32
3 EDB*Loader .......................................................................................................................... 34
   3.1 Data Loading Methods ................................................................................................. 35
   3.2 General Usage ............................................................................................................ 36
   3.3 Building the EDB*Loader Control File ....................................................................... 37
   3.4 Invoking EDB*Loader .................................................................................................. 59
      3.4.1 Exit Codes ............................................................................................................. 65
   3.5 Direct Path Load ......................................................................................................... 65
   3.6 Parallel Direct Path Load ......................................................................................... 66
   3.7 Remote Loading ......................................................................................................... 69
   3.8 Updating a Table with a Conventional Path Load ....................................................... 70
4 EDB*Wrap ........................................................................................................................... 73
   4.1 Using EDB*Wrap to Obfuscate Source Code ............................................................... 74
5 Dynamic Runtime Instrumentation Tools Architecture (DRITA) ........................................ 78
   5.1 Configuring and Using DRITA ...................................................................................... 78
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2</td>
<td>DRITA Functions</td>
<td>80</td>
</tr>
<tr>
<td>5.2.1</td>
<td>get_snaps()</td>
<td>80</td>
</tr>
<tr>
<td>5.2.2</td>
<td>sys_rpt()</td>
<td>80</td>
</tr>
<tr>
<td>5.2.3</td>
<td>sess_rpt()</td>
<td>81</td>
</tr>
<tr>
<td>5.2.4</td>
<td>sessid_rpt()</td>
<td>82</td>
</tr>
<tr>
<td>5.2.5</td>
<td>sesshist_rpt()</td>
<td>84</td>
</tr>
<tr>
<td>5.2.6</td>
<td>purgesnap()</td>
<td>85</td>
</tr>
<tr>
<td>5.2.7</td>
<td>truncsnap()</td>
<td>86</td>
</tr>
<tr>
<td>5.3</td>
<td>Simulating Statspack AWR Reports</td>
<td>87</td>
</tr>
<tr>
<td>5.3.1</td>
<td>edbreport()</td>
<td>87</td>
</tr>
<tr>
<td>5.3.2</td>
<td>stat_db_rpt()</td>
<td>95</td>
</tr>
<tr>
<td>5.3.3</td>
<td>stat_tables_rpt()</td>
<td>96</td>
</tr>
<tr>
<td>5.3.4</td>
<td>statio_tables_rpt()</td>
<td>99</td>
</tr>
<tr>
<td>5.3.5</td>
<td>stat_indexes_rpt()</td>
<td>101</td>
</tr>
<tr>
<td>5.3.6</td>
<td>statio_indexes_rpt()</td>
<td>103</td>
</tr>
<tr>
<td>5.4</td>
<td>Performance Tuning Recommendations</td>
<td>105</td>
</tr>
<tr>
<td>5.5</td>
<td>Event Descriptions</td>
<td>106</td>
</tr>
<tr>
<td>6</td>
<td>Acknowledgements</td>
<td>109</td>
</tr>
</tbody>
</table>
1 Introduction

The tools and utilities documented in this guide allow a developer that is accustomed to working with Oracle utilities to work with Advanced Server in a familiar environment.

The sections in this guide describe compatible tools and utilities that are supported by Advanced Server. These include:

- EDB*Plus
- EDB*Loader
- EDB*Wrap
- Dynamic Runtime Instrumentation

For detailed information about the features supported by Advanced Server, please consult the complete library of Advanced Server guides available at:

http://www.enterprisedb.com/products-services-training/products/documentation
1.1 What’s New

The following database compatibility for Oracle features have been added to Advanced Server 9.5 to create Advanced Server 9.6:

- Advanced Server now supports use of the Postgres password file to initially log into EDB*Plus or to change the connection to EDB*Plus with the CONNECT command. This capability provides additional security in that the user’s password is not exposed when making the connection to EDB*Plus. For more information, see Sections 2.1 and 2.2.6.

- Advanced Server now supports the SET TRIMSPool command in EDB*Plus to trim trailing spaces from each output line in the file specified by the SPOOL command. For more information, see Section 2.2.26.

- Advanced Server now supports additional functionality for EDB*Loader including use of a column name in the field condition of the WHEN clause to control loading of data records into the specified table, the RECORDS DELIMITED BY clause to specify a character other than the default newline character to terminate each record in the data file, the BOUNDFillER clause for preventing the use of a field in the data file for loading a column in a table, but allowing it to be used in an expression for a subsequent column, specification of the length for a data type in the control file’s description of the data record, for example CHAR(3), support of the NULLIF clause to set a column to null if the field condition evaluates to TRUE, and use of a SELECT statement in the expression of a field definition to return a value for a column. For more information, see Section 3.2.

- Advanced Server now supports Dynamic Runtime Instrumentation Tools Architecture (DRITA) access to statistics for lightweight lock wait events. For more information, see Section 5.
1.2 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, etc. For example, `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”.


2 EDB*Plus

EDB*Plus is a utility program that provides a command line user interface to the EDB Postgres Advanced Server. EDB*Plus accepts SQL commands, SPL anonymous blocks, and EDB*Plus commands. EDB*Plus commands are compatible with Oracle SQL*Plus commands and provide various capabilities including:

- Querying certain database objects
- Executing stored procedures
- Formatting output from SQL commands
- Executing batch scripts
- Executing OS commands
- Recording output

The following section describes how to connect to an Advanced Server database using EDB*Plus. The final section provides a summary of the EDB*Plus commands.

2.1 Starting EDB*Plus

To open an EDB*Plus command line, navigate through the Applications (or Start) menu to the Advanced Server menu, to the Run SQL Command Line menu, and select the EDB*Plus option. You can also invoke EDB*Plus from the operating system command line with the following command:

```
edbplus [-S[ILENT] | login | /NOLOG | @scriptfile[.ext] ]
```

- `-SILENT`
  
  If specified, the EDB*Plus sign-on banner is suppressed along with all prompts.

- `login`

  Login information for connecting to the database server and database. `login` takes the following format. (There must be no white space within the login information.)

  ```
  username[/password][@{connectstring | variable} ]
  ```

Where:

`username` is a database username with which to connect to the database.
password is the password associated with the specified username. If a password is not provided, but a password is required for authentication, a password file is used if available. If there is no password file or no entry in the password file with the matching connection parameters, then EDB*Plus will prompt for the password. For more information on using password files and authentication methods, see Section 2.2.6.

connectstring is the database connection string.

variable is a variable defined in the login.sql file that contains a database connection string. The login.sql file can be found in the edbplus subdirectory of the Advanced Server home directory.

host[[:port][/dbname] ]

host is the hostname on which the database server resides. If neither @connectstring nor @variable nor /NOLOG is specified, the default host is assumed to be the localhost. port is the port number receiving connections on the database server. If not specified, the default is 5444. dbname is the name of the database to connect to. If not specified the default is edb.

/NOLOG

Specify /NOLOG to start EDB*Plus without establishing a database connection. SQL commands and EDB*Plus commands that require a database connection cannot be used in this mode. The CONNECT command can be subsequently given to connect to a database after starting EDB*Plus with the /NOLOG option.

scriptfile[.ext ]

scriptfile is the name of a file residing in the current working directory, containing SQL and/or EDB*Plus commands that will be automatically executed after startup of EDB*Plus. ext is the filename extension. If the filename extension is sql, then the .sql extension may be omitted when specifying scriptfile. When creating a script file, always name the file with an extension, otherwise it will not be accessible by EDB*Plus. (EDB*Plus will always assume a .sql extension on filenames that are specified with no extension.)

The following example shows user enterprisedb with password, password, connecting to database edb running on a database server on the localhost at port 5444.

C:\Program Files (x86)\PostgresPlus\9.6AS\edbplus>edbplus
enterprisedb/password
Connected to EnterpriseDB 9.6.0.0 (localhost:5444/edb) AS enterprisedb

EDB*Plus: Release 9.6
The following example shows user `enterprisedb` with password, `password`, connecting to database `edb` running on a database server on the localhost at port 5445.

```
C:\Program Files (x86)\PostgresPlus\9.6AS\edbplus>edbplus enterprisedb/password@localhost:5445/edb
Connected to EnterpriseDB 9.6.0.0 (localhost:5445/edb) AS enterprisedb
EDB*Plus: Release 9.6
Copyright (c) 2008-2015, EnterpriseDB Corporation. All rights reserved.
SQL>
```

Using variable `hr_5445` in the `login.sql` file, the following illustrates how it is used to connect to database `hr` on localhost at port 5445.

```
C:\Program Files (x86)\PostgresPlus\9.6AS\edbplus>edbplus enterprisedb/password@hr_5445
Connected to EnterpriseDB 9.6.0.0 (localhost:5445/hr) AS enterprisedb
EDB*Plus: Release 9.6 (Build 28)
Copyright (c) 2008-2015, EnterpriseDB Corporation. All rights reserved.
SQL>
```

The following is the content of the `login.sql` file used in the previous example.

```
define edb="localhost:5445/edb"
define hr_5445="localhost:5445/hr"
```

The following example executes a script file, `dept_query.sql` after connecting to database `edb` on server localhost at port 5444.

```
C:\Program Files (x86)\PostgresPlus\9.6AS\edbplus>edbplus enterprisedb/password @dept_query
Connected to EnterpriseDB 9.6.0.0 (localhost:5444/edb) AS enterprisedb
SQL> SELECT * FROM dept;
DEPTNO  DNAME        LOC
------- -----------  -------
10      ACCOUNTING   NEW YORK
20      RESEARCH     DALLAS
30      SALES        CHICAGO
40      OPERATIONS   BOSTON
SQL> EXIT
Disconnected from EnterpriseDB Database.
```

The following is the content of file `dept_query.sql` used in the previous example.
2.2 Command Summary

The following sections contains a summary of EDB*Plus commands.

2.2.1 ACCEPT

The ACCEPT command displays a prompt and waits for the user’s keyboard input. The value input by the user is placed in the specified variable.

\[ \text{ACCEPT } \text{variable} \]

The following example creates a new variable named \text{my\_name}, accepts a value of John Smith, then displays the value using the DEFINE command.

```
SQL> ACCEPT my_name
Enter value for my_name: John Smith
SQL> DEFINE my_name
DEFINE MY_NAME = "John Smith"
```

2.2.2 APPEND

APPEND is a line editor command that appends the given text to the end of the current line in the SQL buffer.

\[ \text{APPEND } \text{text} \]

In the following example, a SELECT command is built-in the SQL buffer using the APPEND command. Note that two spaces are placed between the APPEND command and the WHERE clause in order to separate dept and WHERE by one space in the SQL buffer.

```
SQL> APPEND SELECT * FROM dept
SQL> LIST
  1* SELECT * FROM dept
SQL> APPEND WHERE deptno = 10
SQL> LIST
  1* SELECT * FROM dept WHERE deptno = 10
```
2.2.3 CHANGE

CHANGE is a line editor command performs a search-and-replace on the current line in the SQL buffer.

\[ \text{CHANGE } / from / [to/ ] \]

If to/ is specified, the first occurrence of text from in the current line is changed to text to. If to/ is omitted, the first occurrence of text from in the current line is deleted.

The following sequence of commands makes line 3 the current line, then changes the department number in the WHERE clause from 20 to 30.

```
SQL> LIST
1  SELECT empno, ename, job, sal, comm
2  FROM emp
3  WHERE deptno = 20
4* ORDER BY empno
SQL> 3
3* WHERE deptno = 20
SQL> CHANGE /20/30/
3* WHERE deptno = 30
SQL> LIST
1  SELECT empno, ename, job, sal, comm
2  FROM emp
3  WHERE deptno = 30
4* ORDER BY empno
```

2.2.4 CLEAR

The CLEAR command removes the contents of the SQL buffer, deletes all column definitions set with the COLUMN command, or clears the screen.

\[ \text{CLEAR } [\text{ BUFFER } | \text{ SQL } | \text{ COLUMNS } | \text{ SCREEN } ] \]

BUFFER | SQL

Clears the SQL buffer.

COLUMNS

Removes column definitions.

SCREEN

Clears the screen. This is the default if no options are specified.
2.2.5 COLUMN

The COLUMN command controls output formatting. The formatting attributes set by using the COLUMN command remain in effect only for the duration of the current session.

\[
\text{COLUMN } \left[ \begin{array}{c}
\text{column} \\
\{ \text{CLEAR} \} | \\
\{ \text{FORMAT} \} \text{ spec} | \\
\{ \text{HEADING} \} \text{ text} | \\
\{ \text{OFF} \} | \text{ON} \}
\end{array} \right]
\]

If the COLUMN command is specified with no subsequent options, formatting options for current columns in effect for the session are displayed.

If the COLUMN command is followed by a column name, then the column name may be followed by one of the following:

1. No other options
2. CLEAR
3. Any combination of FORMAT, HEADING, and one of OFF or ON

\text{column}

Name of a column in a table to which subsequent column formatting options are to apply. If no other options follow column, then the current column formatting options if any, of column are displayed.

\text{CLEAR}

The CLEAR option reverts all formatting options back to their defaults for column. If the CLEAR option is specified, it must be the only option specified.

\text{spec}

Format specification to be applied to column. For character columns, spec takes the following format:

\text{n}

n is a positive integer that specifies the column width in characters within which to display the data. Data in excess of n will wrap around with the specified column width.
For numeric columns, \textit{spec} is comprised of the following elements.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>Display a leading dollar sign.</td>
</tr>
<tr>
<td>,</td>
<td>Display a comma in the indicated position.</td>
</tr>
<tr>
<td>.</td>
<td>Marks the location of the decimal point.</td>
</tr>
<tr>
<td>0</td>
<td>Display leading zeros.</td>
</tr>
<tr>
<td>9</td>
<td>Number of significant digits to display.</td>
</tr>
</tbody>
</table>

If loss of significant digits occurs due to overflow of the format, then all \#’s are displayed.

\texttt{text}

Text to be used for the column heading of \textit{column}.

\texttt{OFF} | \texttt{ON}

If \texttt{OFF} is specified, formatting options are reverted back to their defaults, but are still available within the session. If \texttt{ON} is specified, the formatting options specified by previous \texttt{COLUMN} commands for \textit{column} within the session are re-activated.

The following example shows the effect of changing the display width of the \textit{job} column.

```
SQL> SET PAGESIZE 9999
SQL> COLUMN job FORMAT A5
SQL> COLUMN job
COLUMN   JOB  ON
FORMAT   A5
wrapped
SQL> SELECT empno, ename, job FROM emp;
EMPNO   ENAME   JOB
------- -------- -----    
7369    SMITH   CLERK  
7499    ALLEN   SALES  MAN
7521    WARD    SALES  MAN
7566    JONES   MANAGER
7654    MARTIN  SALES  MAN
7698    BLAKE   MANAGER
```
The following example applies a format to the sal column.

```sql
SQL> COLUMN sal FORMAT $99,999.00
SQL> COLUMN JOB ON FORMAT A5
   wrapped
SQL> COLUMN sal ON FORMAT $99,999.00
   wrapped
SQL> SELECT empno, ename, job, sal FROM emp;
```

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>JOB</th>
<th>SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>7369</td>
<td>SMITH</td>
<td>CLERK</td>
<td>$800.00</td>
</tr>
<tr>
<td>7499</td>
<td>ALLEN</td>
<td>SALESMAN</td>
<td>$1,600.00</td>
</tr>
<tr>
<td>7521</td>
<td>WARD</td>
<td>SALESMAN</td>
<td>$1,250.00</td>
</tr>
<tr>
<td>7566</td>
<td>JONES</td>
<td>MANAGER</td>
<td>$2,975.00</td>
</tr>
<tr>
<td>7654</td>
<td>MARTIN</td>
<td>SALESMAN</td>
<td>$1,250.00</td>
</tr>
<tr>
<td>7698</td>
<td>BLAKE</td>
<td>MANAGER</td>
<td>$2,850.00</td>
</tr>
<tr>
<td>7782</td>
<td>CLARK</td>
<td>MANAGER</td>
<td>$2,450.00</td>
</tr>
<tr>
<td>7788</td>
<td>SCOTT</td>
<td>ANALYST</td>
<td>$3,000.00</td>
</tr>
</tbody>
</table>
```
2.2.6 CONNECT

Change the database connection to a different user and/or connect to a different database. There must be no white space between any of the parameters following the CONNECT command.

```
CON[NECT] username[/password][@{connectstring | variable } ]
```

Where:

- `username` is a database username with which to connect to the database.
- `password` is the password associated with the specified `username`. If a `password` is not provided, but a password is required for authentication, a search is made for a password file, first in the home directory of the Linux operating system account invoking EDB*Plus (or in the `%APPDATA%\postgresql` directory for Windows) and then at the location specified by the `PGPASSFILE` environment variable. The password file is `.pgpass` on Linux hosts and `pgpass.conf` on Windows hosts. The following is an example on a Windows host:

```
C:\Users\Administrator\AppData\Roaming\postgresql\pgpass.conf
```

If a password file cannot be located, or it does not have an entry matching the EDB*Plus connection parameters, then EDB*Plus will prompt for the password. For more information about password files, see the PostgreSQL core documentation at:

https://www.postgresql.org/docs/9.6/static/libpq-pgpass.html

**Note:** When a password is not required, EDB*Plus does not prompt for a password such as when the `trust` authentication method is specified in the
pg_hba.conf file. For more information about the pg_hba.conf file and authentication methods, see the PostgreSQL core documentation at:


connectstring is the database connection string.

variable is a variable defined in the login.sql file that contains a database connection string. The login.sql file can be found in the edbplus subdirectory of the Advanced Server home directory.

In the following example, the database connection is changed to database edb on the localhost at port 5445 with username, smith.

```
SQL> CONNECT smith/mypassword@localhost:5445/edb
Disconnected from EnterpriseDB Database.
Connected to EnterpriseDB 9.6.0.0 (localhost:5445/edb) AS smith
```

From within the session shown above, the connection is changed to username enterprisedb. Also note that the host defaults to the localhost, the port defaults to 5444 (which is not the same as the port previously used), and the database defaults to edb.

```
SQL> CONNECT enterprisedb/password
Disconnected from EnterpriseDB Database.
Connected to EnterpriseDB 9.6.0.0 (localhost:5444/edb) AS enterprisedb
```

### 2.2.7 DEFINE

The DEFINE command creates or replaces the value of a user variable (also called a substitution variable).

```
DEFINE ] [ variable [ = text ] ]
```

If the DEFINE command is given without any parameters, all current variables and their values are displayed.

If DEFINE variable is given, only variable is displayed with its value.

```
DEFINE variable = text assigns text to variable. text may be optionally enclosed within single or double quotation marks. Quotation marks must be used if text contains space characters.
```

The following example defines two variables, dept and name.

```
SQL> DEFINE dept = 20
SQL> DEFINE name = 'John Smith'
```
SQL> DEFINE
DEFINE EDB = "localhost:5445/edb"
DEFINE DEPT = "20"
DEFINE NAME = "John Smith"

Note: The variable EDB is read from the login.sql file located in the edbplus subdirectory of the Advanced Server home directory.

2.2.8 DEL

DEL is a line editor command that deletes one or more lines from the SQL buffer.

```
DEL [ n | n m | n * | n L[AST ] | * | * n | * L[AST ] | L[AST ] ]
```

The parameters specify which lines are to be deleted from the SQL buffer. Two parameters specify the start and end of a range of lines to be deleted. If the DEL command is given with no parameters, the current line is deleted.

\( n \)

\( n \) is an integer representing the \( n \)th line

\( n m \)

\( n \) and \( m \) are integers where \( m \) is greater than \( n \) representing the \( n \)th through the \( m \)th lines

\( * \)

Current line

\( \text{LAST} \)

Last line

In the following example, the fifth and sixth lines containing columns \text{sal} \ and \text{comm}, respectively, are deleted from the SELECT command in the SQL buffer.

SQL> LIST
1 SELECT
2 empno
3 ,ename
4 ,job
5 ,sal
6 ,comm
2.2.9 DESCRIBE

The DESCRIBE command displays:

- A list of columns, column data types, and column lengths for a table or view
- A list of parameters for a procedure or function
- A list of procedures and functions and their respective parameters for a package.

The DESCRIBE command will also display the structure of the database object referred to by a synonym. The syntax is:

`DESC[RIBE] [ schema.]object`

`schema`

Name of the schema containing the object to be described.

`object`

Name of the table, view, procedure, function, or package to be displayed, or the synonym of an object.

2.2.10 DISCONNECT

The DISCONNECT command closes the current database connection, but does not terminate EDB*Plus.

`DISC[ONNECT ]`
2.2.11  EDIT

The EDIT command invokes an external editor to edit the contents of an operating system file or the SQL buffer.

```
ED[IT ] [ filename[.ext ] ]
```

`filename[.ext ]`

`filename` is the name of the file to open with an external editor. `ext` is the filename extension. If the filename extension is `sql`, then the `.sql` extension may be omitted when specifying `filename`. EDIT always assumes a `.sql` extension on filenames that are specified with no extension. If the filename parameter is omitted from the EDIT command, the contents of the SQL buffer are brought into the editor.

2.2.12  EXECUTE

The EXECUTE command executes an SPL procedure from EDB*Plus.

```
EXEC[UTE ] spl_procedure [ ([ parameters ]) ]
```

`spl_procedure`

The name of the SPL procedure to be executed.

`parameters`

Comma-delimited list of parameters. If there are no parameters, then a pair of empty parentheses may optionally be specified.

2.2.13  EXIT

The EXIT command terminates the EDB*Plus session and returns control to the operating system. QUIT is a synonym for EXIT. Specifying no parameters is equivalent to EXIT SUCCESS COMMIT.

```
{ EXIT | QUIT }
```
[ SUCCESS | FAILURE | WARNING | value | variable ]
[ COMMIT | ROLLBACK ] SUCCESS | FAILURE | WARNING

Returns an operating system dependent return code indicating successful operation, failure, or warning for SUCCESS, FAILURE, and WARNING, respectively. The default is SUCCESS.

value

An integer value that is returned as the return code.

variable

A variable created with the DEFINE command whose value is returned as the return code.

COMMIT | ROLLBACK

If COMMIT is specified, uncommitted updates are committed upon exit. If ROLLBACK is specified, uncommitted updates are rolled back upon exit. The default is COMMIT.

2.2.14 GET

The GET command loads the contents of the given file to the SQL buffer.

GET filename[.ext] [ LIST | NOLIST ]

filename[.ext]

filename is the name of the file to load into the SQL buffer. ext is the filename extension. If the filename extension is sql, then the .sql extension may be omitted when specifying filename. GET always assumes a .sql extension on filenames that are specified with no extension.

LIST | NOLIST

If LIST is specified, the content of the SQL buffer is displayed after the file is loaded. If NOLIST is specified, no listing is displayed. The default is LIST.
2.2.15HELP

The HELP command obtains an index of topics or help on a specific topic. The question mark (?) is synonymous with specifying HELP.

{ HELP | ? } { INDEX | topic }

INDEX

Displays an index of available topics.

topic

The name of a specific topic – e.g., an EDB*Plus command, for which help is desired.

2.2.16HOST

The HOST command executes an operating system command from EDB*Plus.

HOST [os_command]

os_command

The operating system command to be executed. If you do not provide an operating system command, EDB*Plus pauses execution and opens a new shell prompt. When the shell exits, EDB*Plus resumes execution.

2.2.17INPUT

The INPUT line editor command adds a line of text to the SQL buffer after the current line.

INPUT text

The following sequence of INPUT commands constructs a SELECT command.

```
SQL> INPUT SELECT empno, ename, job, sal, comm
SQL> INPUT FROM emp
SQL> INPUT WHERE deptno = 20
SQL> INPUT ORDER BY empno
SQL> LIST
  1  SELECT empno, ename, job, sal, comm
```
2.2.18 LIST

LIST is a line editor command that displays the contents of the SQL buffer.

```
L[IST]  [ n | n m | n * | n L[AST] | * | * n | * L[AST] | L[AST] ]
```

The buffer does not include a history of the EDB*Plus commands.

- \( n \)
  - \( n \) represents the buffer line number.

- \( n m \)
  - \( n m \) displays a list of lines between \( n \) and \( m \).

- \( n * \)
  - \( n * \) displays a list of lines that range between line \( n \) and the current line.

- \( n L[AST] \)
  - \( n L[AST] \) displays a list of lines that range from line \( n \) through the last line in the buffer.

- \( * \)
  - \( * \) displays the current line.

- \( * n \)
  - \( * n \) displays a list of lines that range from the current line through line \( n \).

- \( * L[AST] \)
  - \( * L[AST] \) displays a list of lines that range from the current line through the last line.
LAST displays the last line.

2.2.19 PASSWORD

Use the PASSWORD command to change your database password.

    PASSW[ORD] [user_name]

You must have sufficient privileges to use the PASSWORD command to change another user's password. The following example demonstrates using the PASSWORD command to change the password for a user named acctg:

```
SQL> PASSWORD acctg
Changing password for acctg
   New password:
   New password again:
Password successfully changed.
```

2.2.20 PAUSE

The PAUSE command displays a message, and waits for the user to press ENTER.

    PAU[SE] [optional_text]

optional_text specifies the text that will be displayed to the user. If the optional_text is omitted, Advanced Server will display two blank lines. If you double quote the optional_text string, the quotes will be included in the output.

2.2.21 PRINT

The PRINT command displays the value of a bind variable.

    PRI[NT] [bind_variable_name]

bind_variable_name specifies the name of a bind variable. Omit bind_variable_name to generate a list that includes the values of all bind variables.

2.2.22 PROMPT

The PROMPT command displays a message to the user before continuing.
PRO[MPT] [message_text]

message_text specifies the text displayed to the user. Double quote the string to include quotes in the output.

### 2.2.23 QUIT

The QUIT command terminates the session and returns control to the operating system. QUIT is a synonym for EXIT.

QUIT

[SUCCESS | FAILURE | WARNING | value | sub_variable]

[COMMIT | ROLLBACK]

The default value is QUIT SUCCESS COMMIT.

### 2.2.24 REMARK

Use REMARK to include comments in a script.

REM[ARK] [optional_text]

You may also use the following convention to include a comment:

```sql
/*
 * This is an example of a three line comment.
 */
```

### 2.2.25 SAVE

Use the SAVE command to write the SQL Buffer to an operating system file.

SAVE file_name

[CREATE | REPLACE | APPEND]

file_name
*file_name* specifies the name of the file (including the path) where the buffer contents are written. If you do not provide a file extension, `.sql` is appended to the end of the file name.

**CREATE**

Include the `CREATE` keyword to create a new file. A new file is created *only* if a file with the specified name does not already exist. This is the default.

**REPLACE**

Include the `REPLACE` keyword to specify that Advanced Server should overwrite an existing file.

**APPEND**

Include the `APPEND` keyword to specify that Advanced Server should append the contents of the SQL buffer to the end of the specified file.

The following example saves the contents of the SQL buffer to a file named *example.sql*, located in the `temp` directory:

```
SQL> SAVE C:\example.sql CREATE
File "example.sql" written.
```

### 2.2.26  **SET**

Use the `SET` command to specify a value for a session level variable that controls EDB*Plus* behavior. The following forms of the `SET` command are valid:

**SET AUTOCOMMIT**

Use the `SET AUTOCOMMIT` command to specify COMMIT behavior for Advanced Server transactions.

```
SET AUTO[COMMIT]

{ON | OFF | IMMEDIATE | statement_count}
```

Please note that EDB*Plus* always automatically commits DDL statements.

*ON*
Specify ON to turn AUTOCOMMIT behavior on.

OFF

Specify OFF to turn AUTOCOMMIT behavior off.

IMMEDIATE

IMMEDIATE has the same effect as ON.

statement_count

Include a value for statement_count to instruct EDB*Plus to issue a commit after the specified count of successful SQL statements.

SET COLUMN SEPARATOR

Use the SET COLUMN SEPARATOR command to specify the text that Advanced Server displays between columns.

SET COLSEP column_separator

The default value of column_separator is a single space.

SET ECHO

Use the SET ECHO command to specify if SQL and EDB*Plus script statements should be displayed onscreen as they are executed.

SET ECHO {ON | OFF}

The default value is OFF.

SET FEEDBACK

The SET FEEDBACK command controls the display of interactive information after a SQL statement executes.

SET FEED[BACK] {ON | OFF | row_threshold}

row_threshold

Specify an integer value for row_threshold. Setting row_threshold to 0 is same as setting FEEDBACK to OFF. Setting row_threshold equal 1 effectively sets FEEDBACK to ON.
SET FLUSH

Use the SET FLUSH command to control display buffering.

    SET FLUSH {ON | OFF}

Set FLUSH to OFF to enable display buffering. If you enable buffering, messages bound for the screen may not appear until the script completes. Please note that setting FLUSH to OFF will offer better performance.

Set FLUSH to ON to disable display buffering. If you disable buffering, messages bound for the screen appear immediately.

SET HEADING

Use the SET HEADING variable to specify if Advanced Server should display column headings for SELECT statements.

    SET HEADING {ON | OFF}

SET HEAD SEPARATOR

The SET HEADSEP command sets the new heading separator character used by the COLUMN HEADING command. The default is '|'.

    SET HEADSEP

SET LINESIZE

Use the SET LINESIZE command to specify the width of a line in characters.

    SET LINESIZE width_of_line

width_of_line

The default value of width_of_line is 132.

SET NEWPAGE

Use the SET NEWPAGE command to specify how many blank lines are printed after a page break.

    SET NEWPAGE lines_per_page

lines_per_page
The default value of `lines_per_page` is 1.

**SET NULL**

Use the `SET NULL` command to specify a string that is displayed to the user when a NULL column value is displayed in the output buffer.

```
SET NULL null_string
```

**SET PAGESIZE**

Use the `SET PAGESIZE` command to specify the number of printed lines that fit on a page.

```
SET PAGESIZE line_count
```

Use the `line_count` parameter to specify the number of lines per page.

**SET SQLCASE**

The `SET SQLCASE` command specifies if SQL statements transmitted to the server should be converted to upper or lower case.

```
SET SQLCASE {MIXED | UPPER | LOWER}
```

**UPPER**

Specify `UPPER` to convert the command text to uppercase.

**LOWER**

Specify `LOWER` to convert the command text to lowercase.

**MIXED**

Specify `MIXED` to leave the case of SQL commands unchanged. The default is `MIXED`.

**SET PAUSE**

The `SET PAUSE` command is most useful when included in a script; the command displays a prompt and waits for the user to press `Return`.

```
SET PAUSE {ON | OFF}
```
If **SET PAUSE** is **ON**, the message **Hit ENTER to continue...** will be displayed before each command is executed.

**SET SPACE**

Use the **SET SPACE** command to specify the number of spaces to display between columns:

```
SET SPACE number_of_spaces
```

**SET SQLPROMPT**

Use **SET SQLPROMPT** to set a value for a user-interactive prompt:

```
SET SQL[PROMPT] "prompt"
```

By default, SQLPROMPT is set to "SQL> ".

**SET TERMOUT**

Use the **SET TERMOUT** command to specify if command output should be displayed onscreen.

```
SET TERM[OUT] {ON | OFF}
```

**SET TIMING**

The **SET TIMING** command specifies if Advanced Server should display the execution time for each SQL statement after it is executed.

```
SET TIMI[NG] {ON | OFF}
```

**SET TRIMSPOOL**

Use the **SET TRIMSPOOL** command to remove trailing spaces from each line in the output file specified by the **SPOOL** command.

```
SET TRIM[SPool] {ON | OFF}
```

The default value is **OFF**.

**SET VERIFY**

 Specifies if both the old and new values of a SQL statement are displayed when a substitution variable is encountered.
SET VERIFY { ON | OFF }

### 2.2.27 SHOW

Use the `SHOW` command to display current parameter values.

```
SHOW {ALL | parameter_name}
```

Display the current parameter settings by including the `ALL` keyword:

```
SQL> SHOW ALL
autocommit OFF
colsep " "
define "&"
echo OFF
FEEDBACK ON for 6 row(s).
flush ON
heading ON
headsep "|
linesize 78
newpage 1
null " "
pagesize 14
pause OFF
serveroutput OFF
spool OFF
sqlcase MIXED
sqlprompt "SQL> "
sqlterminator ";;
suffix "\.sql"
termout ON
timing OFF
verify ON
USER is "enterprisedb"
HOST is "localhost"
PORT is "5444"
DATABASE is "edb"
VERSION is "9.6.0.0"
```

Or display a specific parameter setting by including the `parameter_name` in the `SHOW` command:

```
SQL> SHOW VERSION
VERSION is "9.6.0.0"
```

### 2.2.28 SPOOL

The `SPOOL` command sends output from the display to a file.
SP[ool] output_file | OFF

Use the output_file parameter to specify a path name for the output file.

2.2.29 START

Use the START command to run an EDB*Plus script file; START is an alias for @ command.

START script_file

Specify the name of a script file in the script_file parameter.

2.2.30 UNDEFINE

The UNDEFINE command erases a user variable created by the DEFINE command.

UNDEFINE variable_name [ variable_name...]

Use the variable_name parameter to specify the name of a variable or variables.

2.2.31 WHENEVER SQLERROR

The WHENEVER SQLERROR command provides error handling for SQL errors or PL/SQL block errors. The syntax is:

WHENEVER SQLERROR
   {CONTINUE [COMMIT|ROLLBACK|NONE]
   | EXIT [SUCCESS|FAILURE|WARNING|n|sub_variable]
   [COMMIT|ROLLBACK]}

If Advanced Server encounters an error during the execution of a SQL command or PL/SQL block, EDB*Plus performs the action specified in the WHENEVER SQLERROR command:

Include the CONTINUE clause to instruct EDB*Plus to perform the specified action before continuing.
Include the `COMMIT` clause to instruct EDB*Plus to `COMMIT` the current transaction before exiting or continuing.

Include the `ROLLBACK` clause to instruct EDB*Plus to `ROLLBACK` the current transaction before exiting or continuing.

Include the `NONE` clause to instruct EDB*Plus to continue without committing or rolling back the transaction.

Include the `EXIT` clause to instruct EDB*Plus to perform the specified action and exit if it encounters an error.

Use the following options to specify a status code that EDB*Plus will return before exiting:

```
[SUCCESS|FAILURE|WARNING|n|sub_variable]
```

Please note that EDB*Plus supports substitution variables, but does not support bind variables.
3 EDB*Loader

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

EDB*Loader features include:

- Support for the Oracle SQL*Loader data loading methods - conventional path load, direct path load, and parallel direct path load
- Syntax for control file directives compatible with Oracle SQL*Loader
- Input data with delimiter-separated or fixed-width fields
- Bad file for collecting rejected records
- Loading of multiple target tables
- Discard file for collecting records that do not meet the selection criteria of any target table
- Log file for recording the EDB*Loader session and any error messages
- Data loading from standard input and remote loading, particularly useful for large data sources on remote hosts

These features are explained in detail in the following sections.

Note: The following are important version compatibility restrictions between the EDB*Loader client and the database server.

- Invoking EDB*Loader is done using a client program called edbldr, which is used to pass parameters and directive information to the database server. It is strongly recommended that the 9.6 EDB*Loader client (that is, the edbldr program supplied with Advanced Server 9.6) be used to load data only into version 9.6 of the database server. In general, the EDB*Loader client and database server should be the same version.
- It is possible to use a 9.6 EDB*Loader client to load data into a 9.6 database server, but the new 9.6 EDB*Loader features may not be available under those circumstances.
- Use of a 9.6, 9.5, 9.4 or 9.3 EDB*Loader client is not supported for database servers version 9.2 or earlier.
### 3.1 Data Loading Methods

As with Oracle SQL*Loader, EDB*Loader supports three data loading methods:

- Conventional path load
- Direct path load
- Parallel direct path load

Conventional path load is the default method used by EDB*Loader. Basic insert processing is used to add rows to the table.

The advantage of a conventional path load over the other methods is that table constraints and database objects defined on the table such as primary keys, not null constraints, check constraints, unique indexes, foreign key constraints, and triggers are enforced during a conventional path load.

One exception is that the Advanced Server rules defined on the table are not enforced. EDB*Loader can load tables on which rules are defined, but the rules are not executed. As a consequence, partitioned tables implemented using rules cannot be loaded using EDB*Loader.

**Note:** Advanced Server rules are created by the `CREATE RULE` command. Advanced Server rules are not the same database objects as rules and rule sets used in Oracle.

EDB*Loader also supports direct path loads. A direct path load is faster than a conventional path load, but requires the removal of most types of constraints and triggers from the table. See Section 3.5 for information on direct path loads.

Finally, EDB*Loader supports parallel direct path loads. A parallel direct path load provides even greater performance improvement by permitting multiple EDB*Loader sessions to run simultaneously to load a single table. See Section 3.6 for information on parallel direct path loads.
3.2 General Usage

EDB*Loader can load data files with either delimiter-separated or fixed-width fields, in single-byte or multi-byte character sets. The delimiter can be a string consisting of one or more single-byte or multi-byte characters. Data file encoding and the database encoding may be different. Character set conversion of the data file to the database encoding is supported.

Each EDB*Loader session runs as a single, independent transaction. If an error should occur during the EDB*Loader session that aborts the transaction, all changes made during the session are rolled back.

Generally, formatting errors in the data file do not result in an aborted transaction. Instead, the badly formatted records are written to a text file called the bad file. The reason for the error is recorded in the log file.

Records causing database integrity errors do result in an aborted transaction and rollback. As with formatting errors, the record causing the error is written to the bad file and the reason is recorded in the log file.

Note: EDB*Loader differs from Oracle SQL*Loader in that a database integrity error results in a rollback in EDB*Loader. In Oracle SQL*Loader, only the record causing the error is rejected. Records that were previously inserted into the table are retained and loading continues after the rejected record.

The following are examples of types of formatting errors that do not abort the transaction:

- Attempt to load non-numeric value into a numeric column
- Numeric value is too large for a numeric column
- Character value is too long for the maximum length of a character column
- Attempt to load improperly formatted date value into a date column

The following are examples of types of database errors that abort the transaction and result in the rollback of all changes made in the EDB*Loader session:

- Violation of a unique constraint such as a primary key or unique index
- Violation of a referential integrity constraint
- Violation of a check constraint
- Error thrown by a trigger fired as a result of inserting rows
3.3 Building the EDB*Loader Control File

When you invoke EDB*Loader, the list of arguments provided must include the name of a control file. The control file includes the instructions that EDB*Loader uses to load the table (or tables) from the input data file. The control file includes information such as:

- The name of the input data file containing the data to be loaded.
- The name of the table or tables to be loaded from the data file.
- Names of the columns within the table or tables and their corresponding field placement in the data file.
- Specification of whether the data file uses a delimiter string to separate the fields, or if the fields occupy fixed column positions.
- Optional selection criteria to choose which records from the data file to load into a given table.
- The name of the file that will collect illegally formatted records.
- The name of the discard file that will collect records that do not meet the selection criteria of any table.

The syntax for the EDB*Loader control file is as follows:

```
[ OPTIONS (param=value [, param=value ] ...) ]
LOAD DATA
  [ CHARSET charset ]
  [ INFILE '{ data_file | stdin }' ]
  [ BADFILE 'bad_file' ]
  [ DISCARDFILE 'discard_file' ]
  [ { DISCARDMAX | DISCARDS } max_discard_recs ]
  [ INSERT | APPEND | REPLACE | TRUNCATE ]
  [ PRESERVE BLANKS ]
  { INTO TABLE target_table
    [ WHEN field_condition [ AND field_condition ] ... ]
    [ FIELDS TERMINATED BY 'termstring'
      [ OPTIONALLY ENCLOSED BY 'enclstring' ]
    ]
    [ RECORDS DELIMITED BY 'delimstring' ]
    [ TRAILING NULLCOLS ]
    (field_def [, field_def ] ...)
  } ...
```

where `field_def` defines a field in the specified `data_file` that describes the location, data format, or value of the data to be inserted into `column_name` of the `target_table`. The syntax of `field_def` is the following:

```
column_name {
  CONSTANT val |
  FILLER [ POSITION (start:end) ] [ fieldType ] |
  BOUNDFILLER [ POSITION (start:end) ] [ fieldType ] |
  [ POSITION (start:end) ] [ fieldType ]
```
[ NULLIF field_condition [ AND field_condition ] ...]
[ PRESERVE BLANKS ] [ "expr" ]

where fieldtype is one of:

CHAR [(length)] | DATE [(length)] [ "datemask" ] | INTEGER EXTERNAL [(length)] |
FLOAT EXTERNAL [(length)] | DECIMAL EXTERNAL [(length)] | ZONED EXTERNAL [(length)] | ZONED [(precision[,scale])]

Description

The specification of data_file, bad_file, and discard_file may include the full directory path or a relative directory path to the file name. If the file name is specified alone or with a relative directory path, the file is then assumed to exist (in the case of data_file), or is created (in the case of bad_file or discard_file), relative to the current working directory from which edbldr is invoked.

You can include references to environment variables within the EDB*Loader control file when referring to a directory path and/or file name. Environment variable references are formatted differently on Windows systems than on Linux systems:

- On Linux, the format is $ENV_VARIABLE or ${ENV_VARIABLE}
- On Windows, the format is %ENV_VARIABLE%

Where ENV_VARIABLE is the environment variable that is set to the directory path and/or file name.

The EDBLDR_ENV_STYLE environment variable instructs Advanced Server to interpret environment variable references as Windows-styled references or Linux-styled references irregardless of the operating system on which EDB*Loader resides. You can use this environment variable to create portable control files for EDB*Loader.

- On a Windows system, set EDBLDR_ENV_STYLE to linux or unix to instruct Advanced Server to recognize Linux-style references within the control file.
- On a Linux system, set EDBLDR_ENV_STYLE to windows to instruct Advanced Server to recognize Windows-style references within the control file.

The operating system account enterprise db must have read permission on the directory and file specified by data_file.
The operating system account `enterprisedb` must have write permission on the directories where `bad_file` and `discard_file` are to be written.

**Note:** It is suggested that the file names for `data_file`, `bad_file`, and `discard_file` include extensions of `.dat`, `.bad`, and `.dsc`, respectively. If the provided file name does not contain an extension, EDB*Loader assumes the actual file name includes the appropriate aforementioned extension.

If an EDB*Loader session results in data format errors and the BADFILE clause is not specified, nor is the BAD parameter given on the command line when `edbldr` is invoked, a bad file is created with the name `control_file_base.bad` in the current working directory from which `edbldr` is invoked. `control_file_base` is the base name of the control file (that is, the file name without any extension) used in the `edbldr` session.

If all of the following conditions are true, the discard file is not created even if the EDB*Loader session results in discarded records:

- The DISCARDFILE clause for specifying the discard file is not included in the control file.
- The DISCARD parameter for specifying the discard file is not included on the command line.
- The DISCARDMAX clause for specifying the maximum number of discarded records is not included in the control file.
- The DISCARDS clause for specifying the maximum number of discarded records is not included in the control file.
- The DISCARDMAX parameter for specifying the maximum number of discarded records is not included on the command line.

If neither the DISCARDFILE clause nor the DISCARD parameter for explicitly specifying the discard file name are specified, but DISCARDMAX or DISCARDS is specified, then the EDB*Loader session creates a discard file using the data file name with an extension of `.dsc`.

**Note:** There is a distinction between keywords DISCARD and DISCARDS. DISCARD is an EDB*Loader command line parameter used to specify the discard file name (see Section 3.2). DISCARDS is a clause of the LOAD DATA directive that may only appear in the control file. Keywords DISCARDS and DISCARDMAX provide the same functionality of specifying the maximum number of discarded records allowed before terminating the EDB*Loader session. Records loaded into the database before termination of the EDB*Loader session due to exceeding the DISCARDS or DISCARDMAX settings are kept in the database and are not rolled back.
If one of INSERT, APPEND, REPLACE, or TRUNCATE is specified, it establishes the default action of how rows are to be added to target tables. If omitted, the default action is as if INSERT had been specified.

If the FIELDS TERMINATED BY clause is specified, then the POSITION (start:end) clause may not be specified for any field_def. Alternatively, if the FIELDS TERMINATED BY clause is not specified, then every field_def must contain either the POSITION (start:end) clause, the fieldtype(length) clause, or the CONSTANT clause.

Parameters

OPTIONS param=value

Use the OPTIONS clause to specify param=value pairs that represent an EDB*Loader directive. If a parameter is specified in both the OPTIONS clause and on the command line when edbldr is invoked, the command line setting is used.

Specify one or more of the following parameter/value pairs:

DIRECT= { FALSE | TRUE }

If DIRECT is set to TRUE EDB*Loader performs a direct path load instead of a conventional path load. The default value of DIRECT is FALSE.

See Section 3.5 for information on direct path loads.

ERRORS=error_count

error_count specifies the number of errors permitted before aborting the EDB*Loader session. The default is 50.

FREEZE= { FALSE | TRUE }

Set FREEZE to TRUE to indicate that the data should be copied with the rows frozen. A tuple guaranteed to be visible to all current and future transactions is marked as frozen to prevent transaction ID wrap-around. For more information about frozen tuples, see the PostgreSQL core documentation at:

http://www.postgresql.org/docs/9.5/static/routine-vacuuming.html
You must specify a data-loading type of `TRUNCATE` in the control file when using the `FREEZE` option. `FREEZE` is not supported for direct loading.

By default, `FREEZE` is `FALSE`.

`PARALLEL= { FALSE | TRUE }`

Set `PARALLEL` to `TRUE` to indicate that this EDB*Loader session is one of a number of concurrent EDB*Loader sessions participating in a parallel direct path load. The default value of `PARALLEL` is `FALSE`.

When `PARALLEL` is `TRUE`, the `DIRECT` parameter must also be set to `TRUE`. See Section 3.6 for more information about parallel direct path loads.

`ROWS=n`

`n` specifies the number of rows that EDB*Loader will commit before loading the next set of `n` rows.

If EDB*Loader encounters an invalid row during a load (in which the `ROWS` parameter is specified), those rows committed prior to encountering the error will remain in the destination table.

`SKIP=skip_count`

`skip_count` specifies the number of records at the beginning of the input data file that should be skipped before loading begins. The default is 0.

`SKIP_INDEX_MAINTENANCE={ FALSE | TRUE }`

If `SKIP_INDEX_MAINTENANCE` is `TRUE`, index maintenance is not performed as part of a direct path load, and indexes on the loaded table are marked as invalid. The default value of `SKIP_INDEX_MAINTENANCE` is `FALSE`.

Please note: During a parallel direct path load, target table indexes are not updated, and are marked as invalid after the load is complete.

You can use the `REINDEX` command to rebuild an index. For more information about the `REINDEX` command, see the PostgreSQL core documentation available at:

**charset**

Use the `CHARACTERSET` clause to identify the character set encoding of `data_file` where `charset` is the character set name. This clause is required if the data file encoding differs from the control file encoding. (The control file encoding must always be in the encoding of the client where `edbldr` is invoked.)

Examples of `charset` settings are `UTF8`, `SQL_ASCII`, and `SJIS`.

For more information about client to database character set conversion, see the PostgreSQL core documentation available at:

[http://www.postgresql.org/docs/9.5/static/multibyte.html](http://www.postgresql.org/docs/9.5/static/multibyte.html)

**data_file**

File containing the data to be loaded into `target_table`. Each record in the data file corresponds to a row to be inserted into `target_table`.

If an extension is not provided in the file name, EDB*Loader assumes the file has an extension of `.dat`, for example, `mydatafile.dat`.

**Note:** If the `DATA` parameter is specified on the command line when `edbldr` is invoked, the file given by the command line `DATA` parameter is used instead.

If the `INFILE` clause is omitted as well as the command line `DATA` parameter, then the data file name is assumed to be identical to the control file name, but with an extension of `.dat`.

**stdin**

Specify `stdin` (all lowercase letters) if you want to use standard input to pipe the data to be loaded directly to EDB*Loader. This is useful for data sources generating a large number of records to be loaded.

**bad_file**

File that receives `data_file` records that cannot be loaded due to errors.

If an extension is not provided in the file name, EDB*Loader assumes the file has an extension of `.bad`, for example, `mybadfile.bad`.

**Note:** If the `BAD` parameter is specified on the command line when `edbldr` is invoked, the file given by the command line `BAD` parameter is used instead.
**discard_file**

File that receives input data records that are not loaded into any table because none of the selection criteria are met for tables with the `WHEN` clause, and there are no tables without a `WHEN` clause. (All records meet the selection criteria of a table without a `WHEN` clause.)

If an extension is not provided in the file name, EDB*Loader assumes the file has an extension of `.dsc`, for example, `mydiscardfile.dsc`.

**Note:** If the `DISCARD` parameter is specified on the command line when `edbldr` is invoked, the file given by the command line `DISCARD` parameter is used instead.

```
{ DISCARDMAX | DISCARDS } max_discard_recs
```

Maximum number of discarded records that may be encountered from the input data records before terminating the EDB*Loader session. (A discarded record is described in the preceding description of the `discard_file` parameter.) Either keyword `DISCARDMAX` or `DISCARDS` may be used preceding the integer value specified by `max_discard_recs`.

For example, if `max_discard_recs` is 0, then the EDB*Loader session is terminated if and when a first discarded record is encountered. If `max_discard_recs` is 1, then the EDB*Loader session is terminated if and when a second discarded record is encountered.

When the EDB*Loader session is terminated due to exceeding `max_discard_recs`, prior input data records that have been loaded into the database are retained. They are not rolled back.

**INSERT | APPEND | REPLACE | TRUNCATE**

Specifies how data is to be loaded into the target tables. If one of `INSERT`, `APPEND`, `REPLACE`, or `TRUNCATE` is specified, it establishes the default action for all tables, overriding the default of `INSERT`.

**INSERT**

Data is to be loaded into an empty table. EDB*Loader throws an exception and does not load any data if the table is not initially empty.

**Note:** If the table contains rows, the `TRUNCATE` command must be used to empty the table prior to invoking EDB*Loader. EDB*Loader throws an exception if the `DELETE` command is used to empty the table instead of
the **TRUNCATE** command. Oracle SQL*Loader allows the table to be emptied by using either the **DELETE** or **TRUNCATE** command.

**APPEND**

Data is to be added to any existing rows in the table. The table may be initially empty as well.

**REPLACE**

The **REPLACE** keyword and **TRUNCATE** keywords are functionally identical. The table is truncated by EDB*Loader prior to loading the new data.

**Note:** Delete triggers on the table are not fired as a result of the **REPLACE** operation.

**TRUNCATE**

The table is truncated by EDB*Loader prior to loading the new data. Delete triggers on the table are not fired as a result of the **TRUNCATE** operation.

**PRESERVE BLANKS**

For all target tables, retains leading white space when the optional enclosure delimiters are not present and leaves trailing white space intact when fields are specified with a predetermined size. When omitted, the default behavior is to trim leading and trailing white space.

**target_table**

Name of the table into which data is to be loaded. The table name may be schema-qualified (for example, `enterprisedb.emp`). The specified target must not be a view.

**field_condition**

Conditional clause taking the following form:

```
[ ( ) | { (start:end) | column_name } { = | != | <> } 'val' [ ] ]
```
This conditional clause is used for the WHEN clause, which is part of the INTO TABLE target_table clause, and the NULLIF clause, which is part of the field definition denoted as field_def in the syntax diagram.

start and end are positive integers specifying the column positions in data_file that mark the beginning and end of a field that is to be compared with the constant val. The first character in each record begins with a start value of 1.

column_name specifies the name assigned to a field definition of the data file as defined by field_def in the syntax diagram.

Use of either (start:end) or column_name defines the portion of the record in data_file that is to be compared with the value specified by 'val' to evaluate as either true or false.

All characters used in the field_condition text (particularly in the val string) must be valid in the database encoding. (For performing data conversion, EDB*Loader first converts the characters in val string to the database encoding and then to the data file encoding.)

In the WHEN field_condition [ AND field_condition ] clause, if all such conditions evaluate to TRUE for a given record, then EDB*Loader attempts to insert that record into target_table. If the insert operation fails, the record is written to bad_file.

If for a given record, none of the WHEN clauses evaluate to TRUE for all INTO TABLE clauses, the record is written to discard_file, if a discard file was specified for the EDB*Loader session.

See the description of the NULLIF clause in this Parameters list for the effect of field_condition on this clause.

termstring

String of one or more characters that separates each field in data_file. The characters may be single-byte or multi-byte as long as they are valid in the database encoding. Two consecutive appearances of termstring with no intervening character results in the corresponding column set to null.

enclstring

String of one or more characters used to enclose a field value in data_file. The characters may be single-byte or multi-byte as long as they are valid in the
database encoding. Use \texttt{enclstring} on fields where \texttt{termstring} appears as part of the data.

\textit{delimstring}

String of one or more characters that separates each record in \textit{data_file}. The characters may be single-byte or multi-byte as long as they are valid in the database encoding. Two consecutive appearances of \texttt{delimstring} with no intervening character results in no corresponding row loaded into the table. The last record (in other words, the end of the data file) must also be terminated by the \texttt{delimstring} characters, otherwise the final record is not loaded into the table.

\textbf{Note:} The \texttt{RECORDS DELIMITED BY \textquote{delimstring}} clause is not compatible with Oracle databases.

\textbf{TRAILING NULLCOLS}

If \texttt{TRAILING NULLCOLS} is specified, then the columns in the column list for which there is no data in \textit{data_file} for a given record, are set to null when the row is inserted. This applies only to one or more consecutive columns at the end of the column list.

If fields are omitted at the end of a record and \texttt{TRAILING NULLCOLS} is not specified, EDB*Loader assumes the record contains formatting errors and writes it to the bad file.

\textit{column_name}

Name of a column in \textit{target_table} into which a field value defined by \textit{field_def} is to be inserted. If the field definition includes the \texttt{FILLER} or \texttt{BOUNDFILLER} clause, then \textit{column_name} is not required to be the name of a column in the table. It can be any identifier name since the \texttt{FILLER} and \texttt{BOUNDFILLER} clauses prevent the loading of the field data into a table column.

\textbf{CONSTANT \textit{val}}

Specifies a constant that is type-compatible with the column data type to which it is assigned in a field definition. Single or double quotes may enclose \textit{val}. If \textit{val} contains white space, then enclosing quotation marks must be used.

The use of the \textbf{CONSTANT} clause completely determines the value to be assigned to a column in each inserted row. No other clause may appear in the same field definition.
If the TERMINATED BY clause is used to delimit the fields in data_file, there must be no delimited field in data_file corresponding to any field definition with a CONSTANT clause. In other words, EDB*Loader assumes there is no field in data_file for any field definition with a CONSTANT clause.

FILLER

Specifies that the data in the field defined by the field definition is not to be loaded into the associated column if the identifier of the field definition is an actual column name in the table. In such case, the column is set to null. Use of the FILLER or BOUNDFILLER clause is the only circumstance in which the field definition does not have to be identified by an actual column name.

Unlike the BOUNDFILLER clause, an identifier defined with the FILLER clause must not be referenced in a SQL expression. See the discussion of the expr parameter.

BOUNDFILLER

Specifies that the data in the field defined by the field definition is not to be loaded into the associated column if the identifier of the field definition is an actual column name in the table. In such case, the column is set to null. Use of the FILLER or BOUNDFILLER clause is the only circumstance in which the field definition does not have to be identified by an actual column name.

Unlike the FILLER clause, an identifier defined with the BOUNDFILLER clause may be referenced in a SQL expression. See the discussion of the expr parameter.

POSITION (start:end)

Defines the location of the field in a record in a fixed-width field data file. start and end are positive integers. The first character in the record has a start value of 1.

CHAR [(length)] | DATE [(length)] [ "datemask" ] | INTEGER EXTERNAL [(length)] |
FLOAT EXTERNAL [(length)] | DECIMAL EXTERNAL [(length)] |
ZONED EXTERNAL [(length)] | ZONED [(precision[, scale])]

Field type that describes the format of the data field in data_file.

Note: Specification of a field type is optional (for descriptive purposes only) and has no effect on whether or not EDB*Loader successfully inserts the data in the
field into the table column. Successful loading depends upon the compatibility of the column data type and the field value. For example, a column with data type NUMBER(7,2) successfully accepts a field containing 2600, but if the field contains a value such as 26XX, the insertion fails and the record is written to bad_file.

Please note that ZONED data is not human-readable; ZONED data is stored in an internal format where each digit is encoded in a separate nibble/nybble/4-bit field. In each ZONED value, the last byte contains a single digit (in the high-order 4 bits) and the sign (in the low-order 4 bits).

length

Specifies the length of the value to be loaded into the associated column.

If the POSITION (start:end) clause is specified along with a fieldtype(length) clause, then the ending position of the field is overridden by the specified length value. That is, the length of the value to be loaded into the column is determined by the length value beginning at the start position, and not by the end position of the POSITION (start:end) clause. Thus, the value to be loaded into the column may be shorter than the field defined by POSITION (start:end), or it may go beyond the end position depending upon the specified length size.

If the FIELDS TERMINATED BY 'termstring' clause is specified as part of the INTO TABLE clause, and a field definition contains the fieldtype(length) clause, then a record is accepted as long as the specified length values are greater than or equal to the field lengths as determined by the termstring characters enclosing all such fields of the record. If the specified length value is less than a field length as determined by the enclosing termstring characters for any such field, then the record is rejected.

If the FIELDS TERMINATED BY 'termstring' clause is not specified, and the POSITION (start:end) clause is not included with a field containing the fieldtype(length) clause, then the starting position of this field begins with the next character following the ending position of the preceding field. The ending position of the preceding field is either the end of its length value if the preceding field contains the fieldtype(length) clause, or by its end parameter if the field contains the POSITION (start:end) clause without the fieldtype(length) clause.

precision

Use precision to specify the length of the ZONED value.
If the precision value specified for ZONED conflicts with the length calculated by the server based on information provided with the POSITION clause, EDB*Loader will use the value specified for precision.

scale

scale specifies the number of digits to the right of the decimal point in a ZONED value.

datemask

Specifies the ordering and abbreviation of the day, month, and year components of a date field.

Note: If the DATE field type is specified along with a SQL expression for the column, then datemask must be specified after DATE and before the SQL expression. See the following discussion of the expr parameter.

NULLIF field_condition [ AND field_condition ] ...

Note: See the description of field_condition previously listed in this Parameters section for the syntax of field_condition.

If all field conditions evaluate to TRUE, then the column identified by column_name in the field definition is set to null. If any field condition evaluates to FALSE, then the column is set to the appropriate value as would normally occur according to the field definition.

PRESERVE BLANKS

For the column on which this option appears, retains leading white space when the optional enclosure delimiters are not present and leaves trailing white space intact when fields are specified with a predetermined size. When omitted, the default behavior is to trim leading and trailing white space.

expr

A SQL expression returning a scalar value that is type-compatible with the column data type to which it is assigned in a field definition. Double quotes must enclose expr. expr may contain a reference to any column in the field list (except for fields with the FILLER clause) by prefixing the column name by a colon character (:).

expr may also consist of a SQL SELECT statement. If a SELECT statement is used then the following rules must apply: 1) The SELECT statement must be
enclosed within parentheses (SELECT ...). 2) The select list must consist of exactly one expression following the SELECT keyword. 3) The result set must not return more than one row. If no rows are returned, then the returned value of the resulting expression is null. The following is the syntax for use of the SELECT statement:

"(SELECT expr [ FROM table_list [ WHERE condition ] ])")"

Note: Omitting the FROM table_list clause is not compatible with Oracle databases. If no tables need to be specified, use of the FROM DUAL clause is compatible with Oracle databases.

Examples

The following are some examples of control files and their corresponding data files.

Delimiter-Separated Field Data File

The following control file uses a delimiter-separated data file that appends rows to the emp table:

```
LOAD DATA
INFIL 'emp.dat'
BADFILE 'emp.bad'
APPEND
INTO TABLE emp
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
  TRAILING NULLCOLS
  (empno, ename, job, mgr, hiredate, sal, deptno, comm)
```

In the preceding control file, the APPEND clause is used to allow the insertion of additional rows into the emp table.

The following is the corresponding delimiter-separated data file:

```
9101,ROGERS,CLERK,7902,17-DEC-10,1980.00,20
9102,PETERSON,SALESMAN,7698,20-DEC-10,2600.00,30,2300.00
9103,WARREN,SALESMAN,7698,22-DEC-10,5250.00,30,2500.00
9104,"JONES, JR.",MANAGER,7839,02-AFR-09,7975.00,20
```
The use of the `TRAILING NULLCOLS` clause allows the last field supplying the `comm` column to be omitted from the first and last records. The `comm` column is set to null for the rows inserted from these records.

The double quotation mark enclosure character surrounds the value `JONES, JR.` in the last record since the comma delimiter character is part of the field value.

The following query displays the rows added to the table after the EDB*Loader session:

```sql
SELECT * FROM emp WHERE empno > 9100;
```

<table>
<thead>
<tr>
<th>empno</th>
<th>ename</th>
<th>job</th>
<th>mgr</th>
<th>hiredate</th>
<th>sal</th>
<th>comm</th>
<th>deptno</th>
</tr>
</thead>
<tbody>
<tr>
<td>9101</td>
<td>ROGERS</td>
<td>CLERK</td>
<td>7902</td>
<td>17-DEC-10</td>
<td>1980.00</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>9102</td>
<td>PETERSON</td>
<td>SALESMAN</td>
<td>7698</td>
<td>20-DEC-10</td>
<td>2600.00</td>
<td>2300.00</td>
<td>30</td>
</tr>
<tr>
<td>9103</td>
<td>WARREN</td>
<td>SALESMAN</td>
<td>7698</td>
<td>22-DEC-10</td>
<td>5250.00</td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>9104</td>
<td>JONES, JR.</td>
<td>MANAGER</td>
<td>7839</td>
<td>02-APR-09</td>
<td>7975.00</td>
<td></td>
<td>20</td>
</tr>
</tbody>
</table>

(4 rows)

### Fixed-Width Field Data File

The following example is a control file that loads the same rows into the `emp` table, but uses a data file containing fixed-width fields:

```bash
LOAD DATA
INFIL       'emp_fixed.dat'
BADFILE     'emp_fixed.bad'
APPEND
INTO TABLE emp
TRAILING NULLCOLS
  (empno       POSITION (1:4),
  ename       POSITION (5:14),
  job         POSITION (15:23),
  mgr         POSITION (24:27),
  hiredate    POSITION (28:38),
  sal         POSITION (39:46),
  deptno      POSITION (47:48),
  comm        POSITION (49:56)
)
```

In the preceding control file, the `FIELDS TERMINATED BY` and `OPTIONALLY ENCLOSED BY` clauses are absent. Instead, each field now includes the `POSITION` clause.

The following is the corresponding data file containing fixed-width fields:

<table>
<thead>
<tr>
<th>empno</th>
<th>ename</th>
<th>job</th>
<th>mgr</th>
<th>hiredate</th>
<th>sal</th>
<th>comm</th>
<th>deptno</th>
</tr>
</thead>
<tbody>
<tr>
<td>9101</td>
<td>ROGERS</td>
<td>CLERK</td>
<td>7902</td>
<td>17-DEC-10</td>
<td>1980.00</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>9102</td>
<td>PETERSON</td>
<td>SALESMAN</td>
<td>7698</td>
<td>20-DEC-10</td>
<td>2600.00</td>
<td>2300.00</td>
<td>30</td>
</tr>
<tr>
<td>9103</td>
<td>WARREN</td>
<td>SALESMAN</td>
<td>7698</td>
<td>22-DEC-10</td>
<td>5250.00</td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>9104</td>
<td>JONES, JR.</td>
<td>MANAGER</td>
<td>7839</td>
<td>02-APR-09</td>
<td>7975.00</td>
<td></td>
<td>20</td>
</tr>
</tbody>
</table>

(4 rows)

### Single Physical Record Data File – RECORDS DELIMITED BY Clause
The following example is a control file that loads the same rows into the `emp` table, but uses a data file with one physical record. Each individual record that is to be loaded as a row in the table is terminated by the semicolon character (;) specified by the `RECORDS DELIMITED BY` clause.

```sql
LOAD DATA
  INFILE  'emp_recdelim.dat'
  BADFILE 'emp_recdelim.bad'
  APPEND
  INTO TABLE emp
    FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
    RECORDS DELIMITED BY ''
    TRAILING NULLCOLS
    (empno,
     ename,
     job,
     mgr,
     hiredate,
     sal,
     deptno,
     comm)
```

The following is the corresponding data file. The content is a single, physical record in the data file. The record delimiter character is included following the last record (that is, at the end of the file).

```
9101,ROGERS,CLERK,7902,17-DEC-10,1980.00,20,;9102,PETERSON,SALESMAN,7698,20-DEC-10,2600.00,30,2300.00;9103,WARREN,SALESMAN,7698,22-DEC-10,5250.00,30,2500.00;9104,"JONES, JR.",MANAGER,7839,02-APR-09,7975.00,20,;
```

**FILLER Clause**

The following control file illustrates the use of the `FILLER` clause in the data fields for the `sal` and `comm` columns. EDB*Loader ignores the values in these fields and sets the corresponding columns to null.

```sql
LOAD DATA
  INFILE  'emp_fixed.dat'
  BADFILE 'emp_fixed.bad'
  APPEND
  INTO TABLE emp
    TRAILING NULLCOLS
    (empno       POSITION (1:4),
     ename       POSITION (5:14),
     job         POSITION (15:23),
     mgr         POSITION (24:27),
     hiredate    POSITION (28:38),
     sal         FILLER POSITION (39:46),
     deptno      POSITION (47:48),
     comm        FILLER POSITION (49:56)
    )
```
Using the same fixed-width data file as in the prior fixed-width field example, the resulting rows in the table appear as follows:

```sql
SELECT * FROM emp WHERE empno > 9100;
```

<table>
<thead>
<tr>
<th>empno</th>
<th>ename</th>
<th>job</th>
<th>mgr</th>
<th>hiredate</th>
<th>sal</th>
<th>comm</th>
<th>deptno</th>
</tr>
</thead>
<tbody>
<tr>
<td>9101</td>
<td>ROGERS</td>
<td>CLERK</td>
<td>7902</td>
<td>17-DEC-10 00:00:00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9102</td>
<td>PETERSON</td>
<td>SALESMAN</td>
<td>7698</td>
<td>20-DEC-10 00:00:00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9103</td>
<td>WARREN</td>
<td>SALESMAN</td>
<td>7698</td>
<td>22-DEC-10 00:00:00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9104</td>
<td>&quot;JONES, JR.&quot;</td>
<td>MANAGER</td>
<td>7839</td>
<td>02-APR-09 00:00:00</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(4 rows)

**BOUNDFILLER Clause**

The following control file illustrates the use of the BOUNDFILLER clause in the data fields for the job and mgr columns. EDB®Loader ignores the values in these fields and sets the corresponding columns to null in the same manner as the FILLER clause. However, unlike columns with the FILLER clause, columns with the BOUNDFILLER clause are permitted to be used in an expression as shown for column jobdesc.

```
LOAD DATA
INFILE 'emp.dat'
BADFILE 'emp.bad'
APPEND
INTO TABLE empjob
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
TRAILING NULLCOLS
(
    empno, ename, job BOUNDFILLER, mgr BOUNDFILLER, hiredate FILLER, sal FILLER, deptno FILLER, comm FILLER, jobdesc " :job || ' for manager ' || :mgr"
)
```

The following is the delimiter-separated data file used in this example.

```
9101,ROGERS,CLERK,7902,17-DEC-10,1980.00,20
9102,PETERSON,SALESMAN,7698,20-DEC-10,2600.00,30,2300.00
9103,WARREN,SALESMAN,7698,22-DEC-10,5250.00,30,2500.00
9104,"JONES, JR.",MANAGER,7839,02-APR-09,7975.00,20
```

The following table is loaded using the preceding control file and data file.

```
CREATE TABLE empjob
(
    empno NUMBER(4) NOT NULL CONSTRAINT empjob_pk PRIMARY KEY,
    ename VARCHAR2(10),
    job VARCHAR2(9),
    mgr NUMBER(4),
    jobdesc VARCHAR2(25)
);
```
The resulting rows in the table appear as follows:

```
SELECT * FROM empjob;
```

```
empno |   ename    | job   | mgr  |          jobdesc
-------+------------+-------+------+-------------------------------
   9101 | ROGERS     | CLERK | 7902 | CLERK for manager 7902
   9102 | PETERSON   | SALESMAN | 7698 | SALESMAN for manager 7698
   9103 | WARREN     | SALESMAN | 7698 | SALESMAN for manager 7698
   9104 | JONES, JR. | MANAGER | 7839 | MANAGER for manager 7839
```

(4 rows)

Field Types with Length Specification

The following example is a control file that contains the field type clauses with the length specification:

```
LOAD DATA
INFILE        'emp_fixed.dat'
BADFILE     'emp_fixed.bad'
APPEND
INTO TABLE emp
TRAILING NULLCOLS
(
    empno       CHAR(4),
    ename       CHAR(10),
    job         POSITION (15:23) CHAR(9),
    mgr         INTEGER EXTERNAL(4),
    hiredate    DATE(11) "DD-MON-YY",
    sal         DECIMAL EXTERNAL(8),
    deptno      POSITION (47:48),
    comm        POSITION (49:56) DECIMAL EXTERNAL(8)
)
```

Note that the `POSITION` clause and the `fieldtype(length)` clause can be used individually or in combination as long as each field definition contains at least one of the two clauses.

The following is the corresponding data file containing fixed-width fields:

```
9101ROGERS CLERK  790217-DEC-10 1980.0020
9102PETERSON SALESMAN 769820-DEC-10 2600.0030 2300.00
9103WARREN SALESMAN 769822-DEC-10 5250.0030 2500.00
9104JONES, JR.MANAGER  783902-APR-09 7975.0020
```

The resulting rows in the table appear as follows:

```
SELECT * FROM emp WHERE empno > 9100;
```

```
empno |   ename    |   job    | mgr  |      hiredate      |   sal   |  comm   | deptno
-------+------------+---------+------+-------------------+--------+--------+-------
   9101 | ROGERS     | CLERK    | 7902 | 17-DEC-10 00:00:00 | 1980.00 |         |     20
   9102 | PETERSON   | SALESMAN | 7698 | 20-DEC-10 00:00:00 | 2600.00 | 2300.00 |     30
   9103 | WARREN     | SALESMAN | 7698 | 22-DEC-10 00:00:00 | 5250.00 | 2500.00 |     30
   9104 | JONES, JR. | MANAGER  | 7839 | 02-APR-09 00:00:00 | 7975.00 |         |     20
```

(4 rows)
**NULLIF Clause**

The following example uses the NULLIF clause on the sal column to set it to null for employees of job MANAGER as well as on the comm column to set it to null if the employee is not a SALESMAN and is not in department 30. In other words, a comm value is accepted if the employee is a SALESMAN or is a member of department 30.

The following is the control file:

```sql
LOAD DATA
INFILE 'emp_fixed_2.dat'
BADFILE 'emp_fixed_2.bad'
APPEND
INTO TABLE emp
TRAILING NULLCOLS
(
  empno POSITION (1:4),
  ename POSITION (5:14),
  job  POSITION (15:23),
  mgr  POSITION (24:27),
  hiredate POSITION (28:38),
  sal  POSITION (39:46) NULLIF job = 'MANAGER',
  deptno  POSITION (47:48),
  comm  POSITION (49:56) NULLIF job <> 'SALESMAN' AND deptno <> '30'
)
```

The following is the corresponding data file:

```
9101ROGERS  CLERK    790217-DEC-10   1980.0020
9102PETERSON SALESMAN 769820-DEC-10  2600.0030 2300.00
9103WARREN  SALESMAN 769822-DEC-10  5250.0030 2500.00
9104JONES, JR.MANAGER 783902-APR-09  7975.0020
9105ARNOLDS  CLERK    778213-SEP-10  3750.0030  800.00
9106JACKSON  ANALYST 756603-JAN-11  4500.0040 2000.00
9107MAXWELL  SALESMAN 769820-DEC-10  2600.0010 1600.00
```

The resulting rows in the table appear as follows:

```
<table>
<thead>
<tr>
<th>empno</th>
<th>ename</th>
<th>job</th>
<th>NVL(TO_CHAR(sal),'--null--')</th>
<th>NVL(TO_CHAR(comm),'--null--')</th>
<th>deptno</th>
</tr>
</thead>
<tbody>
<tr>
<td>9101</td>
<td>ROGERS</td>
<td>CLERK</td>
<td>1980.00</td>
<td>--null--</td>
<td>20</td>
</tr>
<tr>
<td>9102</td>
<td>PETERSON</td>
<td>SALESMAN</td>
<td>2600.00</td>
<td>2300.00</td>
<td>30</td>
</tr>
<tr>
<td>9103</td>
<td>WARREN</td>
<td>SALESMAN</td>
<td>5250.00</td>
<td>2500.00</td>
<td>30</td>
</tr>
<tr>
<td>9104</td>
<td>JONES, JR.</td>
<td>MANAGER</td>
<td>--null--</td>
<td>--null--</td>
<td>20</td>
</tr>
<tr>
<td>9105</td>
<td>ARNOLDS</td>
<td>CLERK</td>
<td>3750.00</td>
<td>800.00</td>
<td>30</td>
</tr>
<tr>
<td>9106</td>
<td>JACKSON</td>
<td>ANALYST</td>
<td>4500.00</td>
<td>--null--</td>
<td>40</td>
</tr>
<tr>
<td>9107</td>
<td>MAXWELL</td>
<td>SALESMAN</td>
<td>2600.00</td>
<td>1600.00</td>
<td>10</td>
</tr>
</tbody>
</table>
```

Note that the sal column for employee JONES, JR. is null since the job is MANAGER.
The `comm` values from the data file for employees PETERSON, WARREN, ARNOLDS, and MAXWELL are all loaded into the `comm` column of the `emp` table since these employees are either SALESMAN or members of department 30.

The `comm` value of 2000.00 in the data file for employee JACKSON is ignored and the `comm` column of the `emp` table set to null since this employee is neither a SALESMAN nor is a member of department 30.

**SELECT Statement in a Field Expression**

The following example uses a `SELECT` statement in the expression of the field definition to return the value to be loaded into the column.

```sql
LOAD DATA INFILE 'emp_fixed.dat'
BADFILE 'emp_fixed.bad'
APPEND INTO TABLE emp
TRAILING NULLCOLS
{
    empno       POSITION (1:4),
    ename       POSITION (5:14),
    job         POSITION (15:23) 
        "(SELECT dname FROM dept WHERE deptno = :deptno)",
    mgr         POSITION (24:27),
    hiredate    POSITION (28:38),
    sal         POSITION (39:46),
    deptno      POSITION (47:48),
    comm        POSITION (49:56)
}
```

The content of the `dept` table used in the `SELECT` statement is the following:

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

The following is the corresponding data file:

```
9101ROGERS    CLERK    790217-DEC-10   1980.0020
9102PETERSON  SALESMAN 769820-DEC-10  2600.0030  2300.00
9103WARREN    SALESMAN 769822-DEC-10  5250.0030  2500.00
9104JONES, JR.MANAGER  783902-APR-09  7975.0020
```

The resulting rows in the table appear as follows:

```sql
SELECT * FROM emp WHERE empno > 9100;
empno |   ename    |   job    | mgr  |      hiredate      |   sal   |  comm   | deptno
```
Note that the job column contains the value from the dname column of the dept table returned by the SELECT statement instead of the job name from the data file.

**Multiple INTO TABLE Clauses**

The following example illustrates the use of multiple INTO TABLE clauses. For this example, two empty tables are created with the same data definition as the emp table. The following CREATE TABLE commands create these two empty tables, while inserting no rows from the original emp table:

```sql
CREATE TABLE emp_research AS SELECT * FROM emp WHERE deptno = 99;
CREATE TABLE emp_sales AS SELECT * FROM emp WHERE deptno = 99;
```

The following control file contains two INTO TABLE clauses. Also note that there is no APPEND clause so the default operation of INSERT is used, which requires that tables emp_research and emp_sales be empty.

```sql
LOAD DATA
INFIL "emp_multitbl.dat"
BADFILE "emp_multitbl.bad"
DISCARDFILE "emp_multitbl.dsc"
INTO TABLE emp_research
WHEN (47:48) = '20'
TRAILING NULLCOLS
( empno POSITION (1:4),
  ename POSITION (5:14),
  job POSITION (15:23),
  mgr POSITION (24:27),
  hiredate POSITION (28:38),
  sal POSITION (39:46),
  deptno CONSTANT '20',
  comm POSITION (49:56)
)
INTO TABLE emp_sales
WHEN (47:48) = '30'
TRAILING NULLCOLS
( empno POSITION (1:4),
  ename POSITION (5:14),
  job POSITION (15:23),
  mgr POSITION (24:27),
  hiredate POSITION (28:38),
  sal POSITION (39:46),
  deptno CONSTANT '30',
  comm POSITION (49:56) "ROUND(:comm + (:sal * .25), 0)"
)
The **WHEN** clauses specify that when the field designated by columns 47 thru 48 contains 20, the record is inserted into the `emp_research` table and when that same field contains 30, the record is inserted into the `emp_sales` table. If neither condition is true, the record is written to the discard file named `emp_multitbl.dsc`.

The **CONSTANT** clause is given for column `deptno` so the specified constant value is inserted into `deptno` for each record. When the **CONSTANT** clause is used, it must be the only clause in the field definition other than the column name to which the constant value is assigned.

Finally, column `comm` of the `emp_sales` table is assigned a SQL expression. Column names may be referenced in the expression by prefixing the column name with a colon character (:) .

The following is the corresponding data file:

```
9101ROGERS  CLERK    790217-DEC-10  1980.0020
9102PETERSON SALESMAN 769820-DEC-10  2600.0030  2300.00
9103WARREN  SALESMAN  769822-DEC-10  5250.0030  2500.00
9104JONES, JR.MANAGER 783902-APR-09  7975.0020
9105ARNOLDS CLERK    778213-SEP-10  3750.0010
9106JACKSON  ANALYST  756603-JAN-11  4500.0040
```

Since the records for employees ARNOLDS and JACKSON contain 10 and 40 in columns 47 thru 48, which do not satisfy any of the **WHEN** clauses, EDB*Loader writes these two records to the discard file, `emp_multitbl.dsc`, whose content is shown by the following:

```
9105ARNOLDS CLERK    778213-SEP-10  3750.0010
9106JACKSON  ANALYST  756603-JAN-11  4500.0040
```

The following are the rows loaded into the `emp_research` and `emp_sales` tables:

```
SELECT * FROM emp_research;
empno |   ename    |   job   | mgr  |      hiredate      |   sal   | comm | deptno
-------+------------+--------+------|-------------------+--------+-----+-------
  9101  | ROGERS     | CLERK   | 7902 | 17-DEC-10 00:00:00 | 1980.00 |      |  20.00
  9104  | JONES, JR. | MANAGER | 7839 | 02-APR-09 00:00:00 | 7975.00 |      |  20.00
(2 rows)
SELECT * FROM emp_sales;
empno |   ename |   job    | mgr  |      hiredate      |   sal   |  comm   | deptno
-------+--------+---------+------|-------------------+--------+--------+-------
  9102  | PETERSON  | SALESMAN | 7698 | 20-DEC-10 00:00:00 | 2600.00 | 2950.00 |  30.00
  9103  | WARREN    | SALESMAN | 7698 | 22-DEC-10 00:00:00 | 5250.00 | 3813.00 |  30.00
(2 rows)
```
3.4 Invoking EDB*Loader

You must have superuser privileges to run EDB*Loader. Use the following command to invoke EDB*Loader from the command line:

```
edbldr [ -d dbname ] [ -p port ] [ -h host ]
[ USERID={ username/password | username/ | username | / } ]
CONTROL=control_file
[ DATA=data_file ]
[ BAD=bad_file ]
[ DISCARD=discard_file ]
[ DISCARDMAX=max_discard_recs ]
[ LOG=log_file ]
[ PARFILE=param_file ]
[ DIRECT={ FALSE | TRUE } ]
[ FREEZE={ FALSE | TRUE } ]
[ ERRORS=error_count ]
[ PARALLEL={ FALSE | TRUE } ]
[ ROWS=n ]
[ SKIP=skip_count ]
[ SKIP_INDEX_MAINTENANCE={ FALSE | TRUE } ]
[ edb_resource_group=group_name ]
```

Description

If the -d option, the -p option, or the -h option are omitted, the defaults for the database, port, and host are determined according to the same rules as other Advanced Server utility programs such as edb-psql, for example.

Any parameter listed in the preceding syntax diagram except for the -d option, -p option, -h option, and the PARFILE parameter may be specified in a parameter file. The parameter file is specified on the command line when edbldr is invoked using PARFILE=param_file. Some parameters may be specified in the OPTIONS clause in the control file. See the description of the control file in Section 3.3.

The specification of control_file, data_file, bad_file, discard_file, log_file, and param_file may include the full directory path or a relative directory path to the file name. If the file name is specified alone or with a relative directory path, the file is assumed to exist (in the case of control_file, data_file, or param_file), or to be created (in the case of bad_file, discard_file, or log_file) relative to the current working directory from which edbldr is invoked.

Note: The control file must exist in the character set encoding of the client where edbldr is invoked. If the client is in a different encoding than the database encoding, then the PGCLIENTENCODING environment variable must be set on the client to the
client’s encoding prior to invoking edbldr. This must be done to ensure character set conversion is properly done between the client and the database server.

The operating system account used to invoke edbldr must have read permission on the directories and files specified by control_file, data_file, and param_file.

The operating system account enterpriseDB must have write permission on the directories where bad_file, discard_file, and log_file are to be written.

**Note:** It is suggested that the file names for control_file, data_file, bad_file, discard_file, and log_file include extensions of .ctl, .dat, .bad, .dsc, and .log, respectively. If the provided file name does not contain an extension, EDB*Loader assumes the actual file name includes the appropriate aforementioned extension.

**Parameters**

*dbname*

Name of the database containing the tables to be loaded.

*port*

Port number on which the database server is accepting connections.

*host*

IP address of the host on which the database server is running.

USERID={ username/password | username/ | username | / }

EDB*Loader connects to the database with username. username must be a superuser. password is the password for username.

If the USERID parameter is omitted, EDB*Loader prompts for username and password. If USERID=username/ is specified, then EDB*Loader 1) uses the password file specified by environment variable PGPASSFILE if PGPASSFILE is set, or 2) uses the .pgpass password file (pgpass.conf on Windows systems) if PGPASSFILE is not set. If USERID=username is specified, then EDB*Loader prompts for password. If USERID=/ is specified, the connection is attempted using the operating system account as the user name.

**Note:** The Advanced Server connection environment variables PGUSER and PGPASSWORD are ignored by EDB*Loader. See the PostgreSQL core documentation for information on the PGPASSFILE environment variable and the password file.
CONTROL=control_file

control_file specifies the name of the control file containing EDB*Loader directives. If a file extension is not specified, an extension of .ctl is assumed. See Section 3.3 for a description of the control file.

DATA=data_file

data_file specifies the name of the file containing the data to be loaded into the target table. If a file extension is not specified, an extension of .dat is assumed. See Section 3.3 for a description of the data_file.

Note: Specifying a data_file on the command line overrides the INFILE clause specified in the control file.

BAD=bad_file

bad_file specifies the name of a file that receives input data records that cannot be loaded due to errors. See Section 3.3 for a description of the bad_file.

Note: Specifying a bad_file on the command line overrides any BADFILE clause specified in the control file.

DISCARD=discard_file

discard_file is the name of the file that receives input data records that do not meet any table’s selection criteria. See the description of discard_file in Section 3.3.

Note: Specifying a discard_file using the command line DISCARD parameter overrides the DISCARDFILE clause in the control file.

DISCARDMAX=max_discard_recs

max_discard_recs is the maximum number of discarded records that may be encountered from the input data records before terminating the EDB*Loader session. See the description of max_discard_recs in Section 3.3.

Note: Specifying max_discard_recs using the command line DISCARDMAX parameter overrides the DISCARDMAX or DISCARDS clause in the control file.

LOG=log_file

log_file specifies the name of the file in which EDB*Loader records the results of the EDB*Loader session.
If the LOG parameter is omitted, EDB*Loader creates a log file with the name control_file_base.log in the directory from which edbldr is invoked. control_file_base is the base name of the control file used in the EDB*Loader session. The operating system account enterprise must have write permission on the directory where the log file is to be written.

PARFILE=param_file

param_file specifies the name of the file that contains command line parameters for the EDB*Loader session. Any command line parameter listed in this section except for the -d, -p, and -h options, and the PARFILE parameter itself, can be specified in param_file instead of on the command line.

Any parameter given in param_file overrides the same parameter supplied on the command line before the PARFILE option. Any parameter given on the command line that appears after the PARFILE option overrides the same parameter given in param_file.

Note: Unlike other EDB*Loader files, there is no default file name or extension assumed for param_file, though by Oracle SQL*Loader convention, .par is typically used, but not required, as an extension.

DIRECT= { FALSE | TRUE }

If DIRECT is set to TRUE EDB*Loader performs a direct path load instead of a conventional path load. The default value of DIRECT is FALSE.

See Section 3.5 for information on direct path loads.

FREEZE= { FALSE | TRUE }

Set FREEZE to TRUE to indicate that the data should be copied with the rows frozen. A tuple guaranteed to be visible to all current and future transactions is marked as frozen to prevent transaction ID wrap-around. For more information about frozen tuples, see the PostgreSQL core documentation at:

http://www.postgresql.org/docs/9.5/static/routine-vacuuming.html

You must specify a data-loading type of TRUNCATE in the control file when using the FREEZE option. FREEZE is not supported for direct loading.

By default, FREEZE is FALSE.
ERRORS=error_count

$error_count$ specifies the number of errors permitted before aborting the EDB*Loader session. The default is 50.

PARALLEL= { FALSE | TRUE }

Set PARALLEL to TRUE to indicate that this EDB*Loader session is one of a number of concurrent EDB*Loader sessions participating in a parallel direct path load. The default value of PARALLEL is FALSE.

When PARALLEL is TRUE, the DIRECT parameter must also be set to TRUE. See Section 3.6 for more information about parallel direct path loads.

ROWS=$n$

$n$ specifies the number of rows that EDB*Loader will commit before loading the next set of $n$ rows.

SKIP=skip_count

Number of records at the beginning of the input data file that should be skipped before loading begins. The default is 0.

SKIP_INDEX_MAINTENANCE= { FALSE | TRUE }

If set to TRUE, index maintenance is not performed as part of a direct path load, and indexes on the loaded table are marked as invalid. The default value of SKIP_INDEX_MAINTENANCE is FALSE.

Please note: During a parallel direct path load, target table indexes are not updated, and are marked as invalid after the load is complete.

You can use the REINDEX command to rebuild an index. For more information about the REINDEX command, see the PostgreSQL core documentation available at:

http://www.postgresql.org/docs/9.5/static/sql-reindex.html

edb_resource_group=group_name

$group_name$ specifies the name of an EDB Resource Manager resource group to which the EDB*Loader session is to be assigned.
Any default resource group that may have been assigned to the session (for example, a database user running the EDB*Loader session who had been assigned a default resource group with the `ALTER ROLE ... SET edb_resource_group` command) is overridden by the resource group given by the `edb_resource_group` parameter specified on the `edbldr` command line.

**Examples**

In the following example EDB*Loader is invoked using a control file named `emp.ctl` located in the current working directory to load a table in database `edb`:

```bash
$ /opt/PostgresPlus/9.6AS/bin/edbldr -d edb USERID=enterprisedb/password CONTROL=emp.ctl
EDB*Loader: Copyright (c) 2007-2015, EnterpriseDB Corporation.
Successfully loaded (4) records
```

In the following example, EDB*Loader prompts for the user name and password since they are omitted from the command line. In addition, the files for the bad file and log file are specified with the `BAD` and `LOG` command line parameters.

```bash
$ /opt/PostgresPlus/9.6AS/bin/edbldr -d edb CONTROL=emp.ctl BAD=/tmp/emp.bad LOG=/tmp/emp.log
Enter the user name : enterprisedb
Enter the password :
EDB*Loader: Copyright (c) 2007-2015, EnterpriseDB Corporation.
Successfully loaded (4) records
```

The following example runs EDB*Loader with the same parameters as shown in the preceding example, but using a parameter file located in the current working directory. The `SKIP` and `ERRORS` parameters are altered from their defaults in the parameter file as well. The parameter file, `emp.par`, contains the following:

```text
CONTROL=emp.ctl
BAD=/tmp/emp.bad
LOG=/tmp/emp.log
SKIP=1
ERRORS=10
```

EDB*Loader is invoked with the parameter file as shown by the following:

```bash
$ /opt/PostgresPlus/9.6AS/bin/edbldr -d edb PARFILE=emp.par
Enter the user name : enterprisedb
Enter the password :
EDB*Loader: Copyright (c) 2007-2015, EnterpriseDB Corporation.
Successfully loaded (3) records
```
3.4.1 Exit Codes

When EDB*Loader exits, it will return one of the following codes:

<table>
<thead>
<tr>
<th>Exit Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Indicates that all rows loaded successfully.</td>
</tr>
<tr>
<td>1</td>
<td>Indicates that EDB*Loader encountered command line or syntax errors, or aborted the load operation due to an unrecoverable error.</td>
</tr>
<tr>
<td>2</td>
<td>Indicates that the load completed, but some (or all) rows were rejected or discarded.</td>
</tr>
<tr>
<td>3</td>
<td>Indicates that EDB*Loader encountered fatal errors (such as OS errors). This class of errors is equivalent to the FATAL or PANIC severity levels of PostgreSQL errors.</td>
</tr>
</tbody>
</table>

3.5 Direct Path Load

During a direct path load, EDB*Loader writes the data directly to the database pages, which is then synchronized to disk. The insert processing associated with a conventional path load is bypassed, thereby resulting in a performance improvement.
Bypassing insert processing reduces the types of constraints that may exist on the target table. The following types of constraints are permitted on the target table of a direct path load:

- Primary key
- Not null constraints
- Indexes (unique or non-unique)

The restrictions on the target table of a direct path load are the following:

- Triggers are not permitted
- Check constraints are not permitted
- Foreign key constraints on the target table referencing another table are not permitted
- Foreign key constraints on other tables referencing the target table are not permitted
- The table must not be partitioned
- Rules may exist on the target table, but they are not executed

**Note:** Currently, a direct path load in EDB*Loader is more restrictive than in Oracle SQL*Loader. The preceding restrictions do not apply to Oracle SQL*Loader in most cases. The following restrictions apply to a control file used in a direct path load:

- Multiple table loads are not supported. That is, only one `INTO TABLE` clause may be specified in the control file.
- SQL expressions may not be used in the data field definitions of the `INTO TABLE` clause.
- The `FREEZE` option is not supported for direct path loading.

To run a direct path load, add the `DIRECT=TRUE` option as shown by the following example:

```
$ /opt/PostgresPlus/9.6AS/bin/edbldr -d edb USERID=enterprisedb/password
CONTROL=emp.ctl DIRECT=TRUE
```

```
Successfully loaded (4) records
```

### 3.6 Parallel Direct Path Load

The performance of a direct path load can be further improved by distributing the loading process over two or more sessions running concurrently. Each session runs a direct path load into the same table.
Since the same table is loaded from multiple sessions, the input records to be loaded into the table must be divided amongst several data files so that each EDB*Loader session uses its own data file and the same record is not loaded more than once into the table.

The target table of a parallel direct path load is under the same restrictions as a direct path load run in a single session.

The restrictions on the target table of a direct path load are the following:

- Triggers are not permitted
- Check constraints are not permitted
- Foreign key constraints on the target table referencing another table are not permitted
- Foreign key constraints on other tables referencing the target table are not permitted
- The table must not be partitioned
- Rules may exist on the target table, but they are not executed

In addition, the APPEND clause must be specified in the control file used by each EDB*Loader session.

To run a parallel direct path load, run EDB*Loader in a separate session for each participant of the parallel direct path load. Invocation of each such EDB*Loader session must include the DIRECT=TRUE and PARALLEL=TRUE parameters.

Each EDB*Loader session runs as an independent transaction so if one of the parallel sessions aborts and rolls back its changes, the loading done by the other parallel sessions are not affected.

**Note:** In a parallel direct path load, each EDB*Loader session reserves a fixed number of blocks in the target table in a round-robin fashion. Some of the blocks in the last allocated chunk may not be used, and those blocks remain uninitialized. A subsequent use of the VACUUM command on the target table may show warnings regarding these uninitialized blocks such as the following:

```
WARNING:  relation "emp" page 98264 is uninitialized --- fixing
WARNING:  relation "emp" page 98265 is uninitialized --- fixing
WARNING:  relation "emp" page 98266 is uninitialized --- fixing
```

This is an expected behavior and does not indicate data corruption.

Indexes on the target table are not updated during a parallel direct path load and are therefore marked as invalid after the load is complete. You must use the REINDEX command to rebuild the indexes.
The following example shows the use of a parallel direct path load on the `emp` table.

**Note:** If you attempt a parallel direct path load on the sample `emp` table provided with Advanced Server, you must first remove the triggers and constraints referencing the `emp` table. In addition the primary key column, `empno`, was expanded from `NUMBER(4)` to `NUMBER` in this example to allow for the insertion of a larger number of rows.

The following is the control file used in the first session:

```sql
LOAD DATA
INFILE    '/home/user/loader/emp_parallel_1.dat'
APPEND
INTO TABLE emp
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
TRAILING NULLCOLS
{
    empno,
    ename,
    job,
    mgr,
    hiredate,
    sal,
    deptno,
    comm
}
```

The **APPEND** clause must be specified in the control file for a parallel direct path load.

The following shows the invocation of EDB*Loader in the first session. The **DIRECT=TRUE** and **PARALLEL=TRUE** parameters must be specified.

```
$ /opt/PostgresPlus/9.6AS/bin/edbldr -d edb USERID=enterprisedb/password
CONTROL=emp_parallel_1.ctl DIRECT=TRUE PARALLEL=TRUE
WARNING:  index maintenance will be skipped with PARALLEL load
EDB*Loader: Copyright (c) 2007-2015, EnterpriseDB Corporation.
```

The control file used for the second session appears as follows. Note that it is the same as the one used in the first session, but uses a different data file.

```sql
LOAD DATA
INFILE    '/home/user/loader/emp_parallel_2.dat'
APPEND
INTO TABLE emp
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
TRAILING NULLCOLS
{
    empno,
    ename,
    job,
    mgr,
    hiredate,
    sal,
    deptno,
    comm
}
```
The preceding control file is used in a second session as shown by the following:

```
$ /opt/PostgresPlus/9.6AS/bin/edbldr -d edb USERID=enterprisedb/password
CONTROL=emp_parallel_2.ctl DIRECT=TRUE PARALLEL=TRUE
WARNING: index maintenance will be skipped with PARALLEL load
EDB*Loader: Copyright (c) 2007-2015, EnterpriseDB Corporation.
```

EDB*Loader displays the following message in each session when its respective load operation completes:

```
Successfully loaded (10000) records
```

The following query shows that the index on the `emp` table has been marked as **INVALID**:

```
SELECT index_name, status FROM user_indexes WHERE table_name = 'EMP';
```

```
index_name | status
---------- | --------
EMP_PK     | INVALID
```

(1 row)

**Note:** `user_indexes` is the view of indexes compatible with Oracle databases owned by the current user.

Queries on the `emp` table will not utilize the index unless it is rebuilt using the `REINDEX` command as shown by the following:

```
REINDEX INDEX emp_pk;
```

A subsequent query on `user_indexes` shows that the index is now marked as **VALID**:

```
SELECT index_name, status FROM user_indexes WHERE table_name = 'EMP';
```

```
index_name | status
---------- | --------
EMP_PK     | VALID
```

(1 row)

### 3.7 Remote Loading

EDB*Loader supports a feature called *remote loading*. In remote loading, the database containing the table to be loaded is running on a database server on a different host than from where EDB*Loader is invoked with the input data source.

This feature is useful if you have a large amount of data to be loaded, and you do not want to create a large data file on the host running the database server.
In addition, you can use the standard input feature to pipe the data from the data source such as another program or script, directly to EDB*Loader, which then loads the table in the remote database. This bypasses the process of having to create a data file on disk for EDB*Loader.

Performing remote loading along with using standard input requires the following:

- The `edbldr` program must be installed on the client host on which it is to be invoked with the data source for the EDB*Loader session.
- The control file must contain the clause `INFILE 'stdin'` so you can pipe the data directly into EDB*Loader’s standard input. See Section 3.3 for information on the `INFILE` clause and the EDB*Loader control file.
- All files used by EDB*Loader such as the control file, bad file, discard file, and log file must reside on, or are created on, the client host on which `edbldr` is invoked.
- When invoking EDB*Loader, use the `-h` option to specify the IP address of the remote database server. See Section 3.4 for information on invoking EDB*Loader.
- Use the operating system pipe operator (`|`) or input redirection operator (`<`) to supply the input data to EDB*Loader.

The following example loads a database running on a database server at 192.168.1.14 using data piped from a source named `datasource`.

```
datasource | ./edbldr -d edb -h 192.168.1.14 USERID=enterprisedb/password CONTROL=remote.ctl
```

The following is another example of how standard input can be used:

```
./edbldr -d edb -h 192.168.1.14 USERID=enterprisedb/password CONTROL=remote.ctl < datasource
```

### 3.8 Updating a Table with a Conventional Path Load

You can use EDB*Loader with a conventional path load to update the rows within a table, merging new data with the existing data. When you invoke EDB*Loader to perform an update, the server searches the table for an existing row with a matching primary key:

- If the server locates a row with a matching key, it replaces the existing row with the new row.
- If the server does not locate a row with a matching key, it adds the new row to the table.
To use EDB*Loader to update a table, the table must have a primary key. Please note that you cannot use EDB*Loader to UPDATE a partitioned table.

To perform an UPDATE, use the same steps as when performing a conventional path load:

1. Create a data file that contains the rows you wish to UPDATE or INSERT.

2. Define a control file that uses the INFILE keyword to specify the name of the data file. For information about building the EDB*Loader control file, see Section 3.3.

3. Invoke EDB*Loader, specifying the database name, connection information, and the name of the control file. For information about invoking EDB*Loader, see Section 3.4.

The following example uses the emp table that is distributed with the Advanced Server sample data. By default, the table contains:

```
7521, WARD, MANAGER, 7839, 22-FEB-81 00:00:00, 3000.00, 0.00, 30
```

The following control file (emp_update.ctl) specifies the fields in the table in a comma-delimited list. The control file performs an UPDATE on the emp table:

```
LOAD DATA
  INFILE 'emp_update.dat'
  BADFILE 'emp_update.bad'
  DISCARDFILE 'emp_update.dsc'
UPDATE INTO TABLE emp
FIELDS TERMINATED BY ','
(empno, ename, job, mgr, hiredate, sal, comm, deptno)
```

The data that is being updated or inserted is saved in the emp_update.dat file. emp_update.dat contains:

```
7521, WARD, MANAGER, 7839, 22-FEB-81 00:00:00, 3000.00, 0.00, 30
```
Invoke EDB*Loader, specifying the name of the database (edb), the name of a database superuser (and their associated password) and the name of the control file (emp_update.ctl):

```
edblldr -d edb userid=user_name/password control=emp_update.ctl
```

After performing the update, the `emp` table contains:

```sql
edb=# select * from emp;
empno | ename   | job    | mgr  | hiredate     | sal  | comm | deptno
------|---------|--------|------|--------------|------|------|--------
7369  | SMITH   | CLERK  | 7902 | 17-DEC-80    | 800.00 |       | 20
7499  | ALLEN   | SALESMAN | 7698 | 20-FEB-81    | 1600.00 | 300.00 | 30
7521  | WARD    | MANAGER | 7839 | 22-FEB-81    | 3000.00 | 10.00 | 30
7566  | JONES   | MANAGER | 7839 | 02-APR-81    | 3500.00 | 10.00 | 20
7654  | MARTIN  | SALESMAN | 7698 | 28-SEP-81    | 1500.00 | 400.00 | 30
7698  | BLAKE   | MANAGER | 7839 | 01-MAY-81    | 2850.00 |       | 30
7782  | CLARK   | MANAGER | 7839 | 09-JUN-81    | 2450.00 |       | 10
7788  | SCOTT   | ANALYST | 7566 | 19-APR-87    | 3000.00 |       | 20
7839  | KING    | PRESIDENT |      | 17-NOV-81   | 5000.00 |       | 10
7844  | TURNER  | SALESMAN | 7698 | 08-SEP-81    | 1500.00 | 0.00  | 30
7876  | ADAMS   | CLERK   | 7788 | 23-MAY-87    | 1100.00 |       | 20
7900  | JAMES   | CLERK   | 7698 | 03-DEC-81    | 950.00  |       | 30
7902  | FORD    | ANALYST | 7566 | 03-DEC-81    | 3000.00 |       | 20
7903  | BAKER   | SALESMAN | 7521 | 10-JUN-13    | 1800.00 | 500.00 | 20
7904  | MILLS   | SALESMAN | 7839 | 13-JUN-13    | 1800.00 | 500.00 | 20
7934  | MILLER  | CLERK   | 7782 | 23-JAN-82    | 1300.00 |       | 10
(16 rows)
```

The rows containing information for the three employees that are currently in the `emp` table are updated, while rows are added for the new employees (BAKER and MILLS)
4 EDB*Wrap

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

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

If you wrap a `CREATE PACKAGE` statement, you hide the package API from other developers. You may want to wrap the package body, but not the package header so users can see the package prototypes and other public variables that are defined in the package body. To allow users to see what prototypes the package contains, use EDBWrap to obfuscate only the `CREATE PACKAGE BODY` statement in the edbwrap input file, omitting the `CREATE PACKAGE` statement. The package header source will be stored plaintext, while the package body source and package functions and procedures will be stored obfuscated.

Once wrapped, source code and programs cannot be unwrapped or debugged. Reverse engineering is possible, but would be very difficult.

The entire source file is wrapped into one unit. Any `psql` meta-commands included in the wrapped file will not be recognized when the file is executed; executing an obfuscated file that contains a `psql` meta-command will cause a syntax error. `edbwrap` does not validate SQL source code - if the plaintext form contains a syntax error, `edbwrap` will not complain. Instead, the server will report an error and abort the entire file when you try to execute the obfuscated form.
4.1 Using EDB*Wrap to Obfuscate Source Code

EDB*Wrap is a command line utility; it accepts a single input source file, obfuscates the contents and returns a single output file. When you invoke the edbwrap utility, you must provide the name of the file that contains the source code to obfuscate. You may also specify the name of the file where edbwrap will write the obfuscated form of the code. edbwrap offers three different command-line styles. The first style is compatible with Oracle's wrap utility:

```
edbwrap iname=input_file [oname=output_file]
```

The `iname=input_file` argument specifies the name of the input file; if `input_file` does not contain an extension, edbwrap will search for a file named `input_file.sql`

The `oname=output_file` argument (which is optional) specifies the name of the output file; if `output_file` does not contain an extension, edbwrap will append `.plb` to the name.

If you do not specify an output file name, edbwrap writes to a file whose name is derived from the input file name: edbwrap strips the suffix (typically `.sql`) from the input file name and adds `.plb`.

edbwrap offers two other command-line styles that may feel more familiar:

```
edbwrap --iname input_file [--oname output_file]
edbwrap -i input_file [-o output_file]
```

You may mix command-line styles; the rules for deriving input and output file names are identical regardless of which style you use.

Once edbwrap has produced a file that contains obfuscated code, you typically feed that file into the PostgreSQL server using a client application such as `edb-psql`. The server executes the obfuscated code line by line and stores the source code for SPL and PL/pgSQL programs in wrapped form.

In summary, to obfuscate code with EDB*Wrap, you:

1. Create the source code file.
2. Invoke EDB*Wrap to obfuscate the code.
3. Import the file as if it were in plaintext form.

The following sequence demonstrates edbwrap functionality.

First, create the source code for the `list_emp` procedure (in plaintext form):
[bash] cat listemp.sql
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;
/

You can import the list_emp procedure with a client application such as edb-psql:

[bash] edb-psql edb
Welcome to edb-psql 8.4.3.2, the EnterpriseDB interactive terminal.

Type:
\copyright for distribution terms
\h for help with SQL commands
\? for help with edb-psql commands
\g or terminate with semicolon to execute query
\q to quit

ebd=# \l listemp.sql
CREATE PROCEDURE

You can view the plaintext source code (stored in the server) by examining the pg_proc system table:

ebd=# SELECT prosrc FROM pg_proc WHERE proname = 'list_emp';
prosrc
-------------------------------------------------------------------
  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
(1 row)
ebd=# quit
Next, obfuscate the plaintext file with EDB*Wrap:

```bash
[bash] edbwrap -i listemp.sql
EDB*Wrap Utility: Release 8.4.3.2
Copyright (c) 2004-2013 EnterpriseDB Corporation. All Rights Reserved.

Using encoding UTF8 for input.
Processing listemp.sql to listemp.plb

Examining the contents of the output file (listemp.plb) file reveals that the code is obfuscated:

```bash
[bash] cat listemp.plb
$__EDBwrapped__$
UTF8
d+w6DL30RVaGjYMIzkuoSzAQtBw7MhYFulFkBsYHfdJ0rjwBv+hBr1FCyH6j9SgH
movU+hYI+jR+hR2jbxq3sovHKEy2Ip9y3/GckbQgualRhI1GpyWfE0d1tDUpkYRLN
/OUXmk0/P4H6E198sAHevGb0DI0W1+58NJEvxWfDztpb/s5sdx4660qQ
Oz3X3/ghBVqS2JbcxXPMjmrVr6fAXf68ML9mW2H17fNtxcb5kjSzXxfWR2YxZJf
KFNreHblLDTDV1SEC5wE61GlwhYvXO2f22m1R2IFns0Mtf9fWcnBVAes1YjqR00j6+fcr
er/f/efAFh4=
$__EDBwrapped__$
```

You may notice that the second line of the wrapped file contains an encoding name (in this case, the encoding is UTF8). When you obfuscate a file, edbwrap infers the encoding of the input file by examining the locale. For example, if you are running edbwrap while your locale is set to en_US.utf8, edbwrap assumes that the input file is encoded in UTF8. Be sure to examine the output file after running edbwrap; if the locale contained in the wrapped file does not match the encoding of the input file, you should change your locale and rewrap the input file.

You can import the obfuscated code into the PostgreSQL server using the same tools that work with plaintext code:

```bash
[bash] edb-psql edb
Welcome to edb-psql 8.4.3.2, the EnterpriseDB interactive terminal.

Type: \copyright for distribution terms
\h for help with SQL commands
\? for help with edb-psql commands
\g or terminate with semicolon to execute query
\q to quit

edb=# \i listemp.plb
CREATE PROCEDURE

Now, the pg_proc system table contains the obfuscated code:

```bash
edb=# SELECT prosrc FROM pg_proc WHERE proname = 'list_emp';
prosrc
$__EDBwrapped__$
UTF8
d+w6DL30RVaGjYMIzkuoSzAQtBw7MhYFulFkBsYHfdJ0rjwBv+hBr1FCyH6j9SgH
movU+hYI+jR+hR2jbxq3sovHKEy2Ip9y3/GckbQgualRhI1GpyWfE0d1tDUpkYRLN
/OUXmk0/P4H6E198sAHevGb0DI0W1+58NJEvxWfDztpb/s5sdx4660qQ
Oz3X3/ghBVqS2JbcxXPMjmrVr6fAXf68ML9mW2H17fNtxcb5kjSzXxfWR2YxZJf
KFNreHblLDTDV1SEC5wE61GlwhYvXO2f22m1R2IFns0Mtf9fWcnBVAes1YjqR00j6+fcr
er/f/efAFh4=
$__EDBwrapped__$
```

Invoke the obfuscated code in the same way that you would invoke the plaintext form:

```sql
edb=# exec list_emp;
```

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>7369</td>
<td>SMITH</td>
</tr>
<tr>
<td>7499</td>
<td>ALLEN</td>
</tr>
<tr>
<td>7521</td>
<td>WARD</td>
</tr>
<tr>
<td>7566</td>
<td>JONES</td>
</tr>
<tr>
<td>7654</td>
<td>MARTIN</td>
</tr>
<tr>
<td>7698</td>
<td>BLAKE</td>
</tr>
<tr>
<td>7782</td>
<td>CLARK</td>
</tr>
<tr>
<td>7788</td>
<td>SCOTT</td>
</tr>
<tr>
<td>7839</td>
<td>KING</td>
</tr>
<tr>
<td>7844</td>
<td>TURNER</td>
</tr>
<tr>
<td>7876</td>
<td>ADAMS</td>
</tr>
<tr>
<td>7900</td>
<td>JAMES</td>
</tr>
<tr>
<td>7902</td>
<td>FORD</td>
</tr>
<tr>
<td>7934</td>
<td>MILLER</td>
</tr>
</tbody>
</table>

EDB-SPL Procedure successfully completed

```sql
edb=# quit
```

When you use `pg_dump` to back up a database, wrapped programs remain obfuscated in the archive file.

Be aware that audit logs produced by the Postgres server will show wrapped programs in plaintext form. Source code is also displayed in plaintext in SQL error messages generated during the execution of a program.

Note: At this time, the bodies of the objects created by the following statements will not be stored in obfuscated form:

```sql
CREATE [OR REPLACE] TYPE type_name AS OBJECT
CREATE [OR REPLACE] TYPE type_name UNDER type_name
CREATE [OR REPLACE] TYPE BODY type_name
```
5 Dynamic Runtime Instrumentation Tools Architecture (DRITA)

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

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

5.1 Configuring and Using DRITA

Advanced Server's postgresql.conf file includes a configuration parameter named timed_statistics that controls the collection of timing data. The valid parameter values are TRUE or FALSE; the default value is FALSE.

This is a dynamic parameter which can be modified in the postgresql.conf file, or while a session is in progress. To enable DRITA, you must either:

Modify the postgresql.conf file, setting the timed_statistics parameter to TRUE.

or

Connect to the server with the EDB-PSQL client, and invoke the command:

```
SET timed_statistics = TRUE
```

After modifying the timed_statistics parameter, take a starting snapshot. A snapshot captures the current state of each timer and event counter. The server will compare the starting snapshot to a later snapshot to gauge system performance.

Use the edbsnap() function to take the beginning snapshot:

```
edb=# SELECT * FROM edbsnap();
edbsnap
----------------------
 Statement processed.  
(1 row)
```
Then, run the workload that you would like to evaluate; when the workload has completed (or at a strategic point during the workload), take another snapshot:

```
edb=# SELECT * FROM edbsnap();
   edbsnap
----------------------
    Statement processed.
   (1 row)
```

You can capture multiple snapshots during a session. Then, use the DRITA functions and reports to manage and compare the snapshots to evaluate performance information.
5.2 **DRITA Functions**

You can use DRITA functions to gather wait information and manage snapshots. DRITA functions are fully supported by Advanced Server 9.6 whether your installation is made compatible with Oracle databases or is made in PostgreSQL-compatible mode.

### 5.2.1 `get_snaps()`

The `get_snaps()` function returns a list of the current snapshots. The signature is:

```sql
get_snaps()
```

The following example demonstrates using the `get_snaps()` function to display a list of snapshots:

```sql
edb=# SELECT * FROM get_snaps();
get_snaps
----------------
  1 11-FEB-10  10:41:05.668852
  2 11-FEB-10  10:42:27.26154
  3 11-FEB-10  10:45:48.999992
  4 11-FEB-10  11:01:58.345163
  5 11-FEB-10  11:05:14.092683
  6 11-FEB-10  11:06:33.151002
  7 11-FEB-10  11:11:16.405664
  8 11-FEB-10  11:13:29.458405
  9 11-FEB-10  11:23:57.595916
 10 11-FEB-10  11:29:02.214014
 11 11-FEB-10  11:31:44.244038
(11 rows)
```

The first column in the result list displays the snapshot identifier; the second column displays the date and time that the snapshot was captured.

### 5.2.2 `sys_rpt()`

The `sys_rpt()` function returns system wait information. The signature is:

```sql
sys_rpt(beginning_id, ending_id, top_n)
```

**Parameters**

- `beginning_id`

  `beginning_id` is an integer value that represents the beginning session identifier.
**ending_id**

*ending_id* is an integer value that represents the ending session identifier.

**top_n**

*top_n* represents the number of rows to return

This example demonstrates a call to the `sys_rpt()` function:

```
edb=# SELECT * FROM sys_rpt(9, 10, 10);
```

<table>
<thead>
<tr>
<th>WAIT NAME</th>
<th>COUNT</th>
<th>WAIT TIME</th>
<th>% WAIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>wal write</td>
<td>21250</td>
<td>104.723772</td>
<td>36.31</td>
</tr>
<tr>
<td>db file read</td>
<td>121407</td>
<td>72.143274</td>
<td>25.01</td>
</tr>
<tr>
<td>wal flush</td>
<td>84185</td>
<td>51.652495</td>
<td>17.91</td>
</tr>
<tr>
<td>wal file sync</td>
<td>712</td>
<td>29.482206</td>
<td>10.22</td>
</tr>
<tr>
<td>infinitecache write</td>
<td>84178</td>
<td>15.814444</td>
<td>5.48</td>
</tr>
<tr>
<td>db file write</td>
<td>84177</td>
<td>14.447718</td>
<td>5.01</td>
</tr>
<tr>
<td>infinitecache read</td>
<td>672</td>
<td>0.098691</td>
<td>0.03</td>
</tr>
<tr>
<td>db file extend</td>
<td>190</td>
<td>0.040386</td>
<td>0.01</td>
</tr>
<tr>
<td>query plan</td>
<td>52</td>
<td>0.024400</td>
<td>0.01</td>
</tr>
<tr>
<td>wal insert lock acquire</td>
<td>4</td>
<td>0.000837</td>
<td>0.00</td>
</tr>
</tbody>
</table>

(12 rows)

The information displayed in the result set includes:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAIT NAME</td>
<td>The name of the wait.</td>
</tr>
<tr>
<td>COUNT</td>
<td>The number of times that the wait event occurred.</td>
</tr>
<tr>
<td>WAIT TIME</td>
<td>The time of the wait event in milliseconds.</td>
</tr>
<tr>
<td>% WAIT</td>
<td>The percentage of the total wait time used by this wait for this session.</td>
</tr>
</tbody>
</table>

### 5.2.3 sess_rpt()

The `sess_rpt()` function returns session wait information. The signature is:

```
sess_rpt(beginning_id, ending_id, top_n)
```

**Parameters**

*beginning_id*
`beginning_id` is an integer value that represents the beginning session identifier.

`ending_id`

`ending_id` is an integer value that represents the ending session identifier.

`top_n`

`top_n` represents the number of rows to return.

The following example demonstrates a call to the `sess_rpt()` function:

```sql
SELECT * FROM sess_rpt(18, 19, 10);
```

The information displayed in the result set includes:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>The processID of the session.</td>
</tr>
<tr>
<td>USER</td>
<td>The name of the user incurring the wait.</td>
</tr>
<tr>
<td>WAIT_NAME</td>
<td>The name of the wait event.</td>
</tr>
<tr>
<td>COUNT</td>
<td>The number of times that the wait event occurred.</td>
</tr>
<tr>
<td>TIME (ms)</td>
<td>The length of the wait event in milliseconds.</td>
</tr>
<tr>
<td>% WAIT SES</td>
<td>The percentage of the total wait time used by this wait for this session.</td>
</tr>
<tr>
<td>% WAIT ALL</td>
<td>The percentage of the total wait time used by this wait (for all sessions).</td>
</tr>
</tbody>
</table>

5.2.4 `sessid_rpt()`

The `sessid_rpt()` function returns session ID information for a specified backend. The signature is:
`sessid_rpt(beginning_id, ending_id, backend_id)`

**Parameters**

**beginning_id**

*beginning_id* is an integer value that represents the beginning session identifier.

**ending_id**

*ending_id* is an integer value that represents the ending session identifier.

**backend_id**

*backend_id* is an integer value that represents the backend identifier.

The following code sample demonstrates a call to `sessid_rpt()`:

```sql
SELECT * FROM sessid_rpt(18, 19, 17373);
```

<table>
<thead>
<tr>
<th>ID</th>
<th>USER</th>
<th>WAIT NAME</th>
<th>COUNT</th>
<th>TIME (ms)</th>
<th>% WAIT SES</th>
<th>% WAIT ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>17373</td>
<td>enterprise db file read</td>
<td>30 0.175713 85.24</td>
<td>85.24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17373</td>
<td>enterprise query plan</td>
<td>18 0.014930 7.24</td>
<td>7.24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17373</td>
<td>enterprise wal flush</td>
<td>6  0.004067 1.97</td>
<td>1.97</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17373</td>
<td>enterprise wal write</td>
<td>1  0.004063 1.97</td>
<td>1.97</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17373</td>
<td>enterprise wal file sync</td>
<td>1 0.003664 1.78</td>
<td>1.78</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17373</td>
<td>enterprise infinitecache read</td>
<td>38 0.003076 1.49</td>
<td>1.49</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17373</td>
<td>enterprise infinitecache write</td>
<td>5 0.000548 0.27</td>
<td>0.27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17373</td>
<td>enterprise db file extend</td>
<td>190 0.040386 0.03</td>
<td>0.03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17373</td>
<td>enterprise db file write</td>
<td>5 0.000082 0.04</td>
<td>0.04</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(11 rows)

The information displayed in the result set includes:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>The process ID of the wait.</td>
</tr>
<tr>
<td>USER</td>
<td>The name of the user that owns the session.</td>
</tr>
<tr>
<td>WAIT NAME</td>
<td>The name of the wait event.</td>
</tr>
<tr>
<td>COUNT</td>
<td>The number of times that the wait event occurred.</td>
</tr>
<tr>
<td>TIME (ms)</td>
<td>The length of the wait in milliseconds.</td>
</tr>
<tr>
<td>% WAIT SES</td>
<td>The percentage of the total wait time used by this wait for this session.</td>
</tr>
<tr>
<td>% WAIT ALL</td>
<td>The percentage of the total wait time used by this wait (for all sessions).</td>
</tr>
</tbody>
</table>
5.2.5 `sesshist_rpt()`

The `sesshist_rpt()` function returns session wait information for a specified backend. The signature is:

```
(sesshist_rpt(snapshot_id, session_id)
```

**Parameters**

`snapshot_id`

`snapshot_id` is an integer value that identifies the snapshot.

`session_id`

`session_id` is an integer value that represents the session.

The following example demonstrates a call to the `sesshist_rpt()` function:

```sql
edb=# SELECT * FROM sesshist_rpt (9, 5531);
```

<table>
<thead>
<tr>
<th>ID</th>
<th>USER</th>
<th>SEQ</th>
<th>WAIT NAME</th>
<th>ELAPSED(ms)</th>
<th>File</th>
<th>Name</th>
<th># of Blk</th>
<th>Sum of Blks</th>
</tr>
</thead>
<tbody>
<tr>
<td>5531</td>
<td>enterprise 1</td>
<td>18546</td>
<td>db file read</td>
<td></td>
<td>14309</td>
<td>session_waits_pk</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5531</td>
<td>enterprise 2</td>
<td>125</td>
<td>infinitecache read</td>
<td></td>
<td>14309</td>
<td>session_waits_pk</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5531</td>
<td>enterprise 3</td>
<td>376</td>
<td>db file read</td>
<td></td>
<td>14304</td>
<td>edb$session_waits</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5531</td>
<td>enterprise 4</td>
<td>166</td>
<td>infinitecache read</td>
<td></td>
<td>14304</td>
<td>edb$session_waits</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5531</td>
<td>enterprise 5</td>
<td>7978</td>
<td>db file read</td>
<td></td>
<td>1260</td>
<td>pg_authid</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5531</td>
<td>enterprise 6</td>
<td>154</td>
<td>infinitecache read</td>
<td></td>
<td>1260</td>
<td>pg_authid</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5531</td>
<td>enterprise 7</td>
<td>628</td>
<td>db file read</td>
<td></td>
<td>14302</td>
<td>system_waitsPk</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5531</td>
<td>enterprise 8</td>
<td>463</td>
<td>infinitecache read</td>
<td></td>
<td>14302</td>
<td>system_waitsPk</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5531</td>
<td>enterprise 9</td>
<td>3446</td>
<td>db file read</td>
<td></td>
<td>14297</td>
<td>edb$system_waitsPk</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5531</td>
<td>enterprise 10</td>
<td>187</td>
<td>infinitecache read</td>
<td></td>
<td>14297</td>
<td>edb$system_waitsPk</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5531</td>
<td>enterprise 11</td>
<td>14750</td>
<td>db file read</td>
<td></td>
<td>14295</td>
<td>snap_pk</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
The information displayed in the result set includes:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>The system-assigned identifier of the wait.</td>
</tr>
<tr>
<td>USER</td>
<td>The name of the user that incurred the wait.</td>
</tr>
<tr>
<td>SEQ</td>
<td>The sequence number of the wait event.</td>
</tr>
<tr>
<td>WAIT_NAME</td>
<td>The name of the wait event.</td>
</tr>
<tr>
<td>ELAPSED (ms)</td>
<td>The length of the wait event in milliseconds.</td>
</tr>
<tr>
<td>File</td>
<td>The relfilenode number of the file.</td>
</tr>
<tr>
<td>Name</td>
<td>If available, the name of the file name related to the wait event.</td>
</tr>
<tr>
<td># of Blk</td>
<td>The block number read or written for a specific instance of the event.</td>
</tr>
<tr>
<td>Sum of Blks</td>
<td>The number of blocks read.</td>
</tr>
</tbody>
</table>

### 5.2.6 purgesnap()

The `purgesnap()` function purges a range of snapshots from the snapshot tables. The signature is:

```
purgesnap(beginning_id, ending_id)
```

**Parameters**

`beginning_id`

`beginning_id` is an integer value that represents the beginning session identifier.

`ending_id`

`ending_id` is an integer value that represents the ending session identifier.

`purgesnap()` removes all snapshots between `beginning_id` and `ending_id` (inclusive):

```
SELECT * FROM purgesnap(6, 9);
```
A call to the `get_snaps()` function after executing the example shows that snapshots 6 through 9 have been purged from the snapshot tables:

```
edb=# SELECT * FROM get_snaps();
get_snaps
-------------------------------
    get_snaps                   
1  11-FEB-10 10:41:05.668852
2  11-FEB-10 10:42:27.26154
3  11-FEB-10 10:45:48.999992
4  11-FEB-10 11:01:58.345163
5  11-FEB-10 11:05:14.092683
10 11-FEB-10 11:29:02.214014
11 11-FEB-10 11:31:44.244038
(7 rows)
```

### 5.2.7 `truncsnap()`

Use the `truncsnap()` function to delete all records from the snapshot table. The signature is:

```
truncsnap()
```

For example:

```
SELECT * FROM truncsnap();
truncsnap
------------------------
Snapshots truncated.
(1 row)
```

A call to the `get_snaps()` function after calling the `truncsnap()` function shows that all records have been removed from the snapshot tables:

```
SELECT * FROM get_snaps();
get_snaps
----------
(0 rows)
```
5.3 Simulating Statspack AWR Reports

The functions described in this section return information comparable to the information contained in an Oracle Statspack/AWR (Automatic Workload Repository) report. When taking a snapshot, performance data from system catalog tables is saved into history tables. The reporting functions listed below report on the differences between two given snapshots.

- stat_db_rpt()
- stat_tables_rpt()
- statio_tables_rpt()
- stat_indexes_rpt()
- statio_indexes_rpt()

The reporting functions can be executed individually or you can execute all five functions by calling the edbreport() function.

5.3.1 edbreport()

The edbreport() function includes data from the other reporting functions, plus additional system information. The signature is:

    edbreport(beginning_id, ending_id)

Parameters

beginning_id

    beginning_id is an integer value that represents the beginning session identifier.

ending_id

    ending_id is an integer value that represents the ending session identifier.

The call to the edbreport() function returns a composite report that contains system information and the reports returned by the other statspack functions. :

edb=# SELECT * FROM edbreport(9, 10);

edbreport
------------------------------------------------------------------------------
EnterpriseDB Report for database edb        23-AUG-15
Version: EnterpriseDB 9.6.0.0 on i686-pc-linux-gnu
Begin snapshot: 9 at 23-AUG-15 13:45:07.165123
The information displayed in the report introduction includes the database name and version, the current date, the beginning and ending snapshot date and times, database and tablespace details and schema information.

The information displayed in the **Top 10 Relations by pages** section includes:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>RELPAGES</td>
<td>The number of pages in the table.</td>
</tr>
</tbody>
</table>

The information displayed in the **Top 10 Indexes by pages** section includes:
The information displayed in the Top 10 Indexes by pages section includes:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDEX</td>
<td>The name of the index.</td>
</tr>
<tr>
<td>RELPAGES</td>
<td>The number of pages in the index.</td>
</tr>
</tbody>
</table>

Top 10 Relations by DML

<table>
<thead>
<tr>
<th>SCHEMA</th>
<th>RELATION</th>
<th>UPDATES</th>
<th>DELETES</th>
<th>INSERTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>enterprisesdb</td>
<td>pgbench_accounts</td>
<td>10400</td>
<td>0</td>
<td>1000000</td>
</tr>
<tr>
<td>enterprisesdb</td>
<td>pgbench_tellers</td>
<td>10400</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>enterprisesdb</td>
<td>pgbench_branches</td>
<td>10400</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>enterprisesdb</td>
<td>pgbench_history</td>
<td>0</td>
<td>0</td>
<td>10400</td>
</tr>
<tr>
<td>pgagent</td>
<td>pga_jobclass</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>pgagent</td>
<td>pga_exception</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pgagent</td>
<td>pga_job</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pgagent</td>
<td>pga_jobagent</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pgagent</td>
<td>pga_joblog</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pgagent</td>
<td>pga_jobstep</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The information displayed in the Top 10 Relations by DML section includes:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA</td>
<td>The name of the schema in which the table resides.</td>
</tr>
<tr>
<td>RELATION</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>UPDATES</td>
<td>The number of UPDATES performed on the table.</td>
</tr>
<tr>
<td>DELETES</td>
<td>The number of DELETES performed on the table.</td>
</tr>
<tr>
<td>INSERTS</td>
<td>The number of INSERTS performed on the table.</td>
</tr>
</tbody>
</table>

The information displayed in the DATA from pg_stat_database section of the report includes:

<table>
<thead>
<tr>
<th>DATABASE</th>
<th>NUMBACKENDS</th>
<th>XACT COMMIT</th>
<th>XACT ROLLBACK</th>
<th>BLKS READ</th>
<th>BLKS HIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>edb</td>
<td>0</td>
<td>142</td>
<td>0</td>
<td>78</td>
<td>10446</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>100.26</td>
<td>0.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The information displayed in the DATA from pg_stat_database section of the report includes:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATABASE</td>
<td>The name of the database.</td>
</tr>
<tr>
<td>NUMBACKENDS</td>
<td>Number of backends currently connected to this database. This is the only column in this view that returns a value reflecting current state; all other columns return the accumulated values since the last reset.</td>
</tr>
<tr>
<td>XACT COMMIT</td>
<td>Number of transactions in this database that have been committed.</td>
</tr>
<tr>
<td>XACT ROLLBACK</td>
<td>Number of transactions in this database that have been rolled back.</td>
</tr>
<tr>
<td>BLKS READ</td>
<td>Number of disk blocks read in this database.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BLKS HIT</td>
<td>Number of times disk blocks were found already in the buffer cache (when a read was not necessary).</td>
</tr>
<tr>
<td>BLKS ICACHE HIT</td>
<td>The number of blocks found in Infinite Cache.</td>
</tr>
<tr>
<td>HIT RATIO</td>
<td>The percentage of times that a block was found in the shared buffer cache.</td>
</tr>
<tr>
<td>ICACHE HIT RATIO</td>
<td>The percentage of times that a block was found in Infinite Cache.</td>
</tr>
</tbody>
</table>

**DATA from pg_buffercache**

<table>
<thead>
<tr>
<th>RELATION</th>
<th>BUFFERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>pgbench_accounts</td>
<td>16671</td>
</tr>
<tr>
<td>pgbench_accounts_pkey</td>
<td>2745</td>
</tr>
<tr>
<td>pgbench_history</td>
<td>590</td>
</tr>
<tr>
<td>pg_statistic</td>
<td>39</td>
</tr>
<tr>
<td>edb$statio_all_indexes</td>
<td>31</td>
</tr>
<tr>
<td>edb$stat_all_indexes</td>
<td>31</td>
</tr>
<tr>
<td>edb$statio_all_tables</td>
<td>21</td>
</tr>
<tr>
<td>edb$stat_all_tables</td>
<td>20</td>
</tr>
<tr>
<td>pg_depend</td>
<td>20</td>
</tr>
<tr>
<td>pg_operator</td>
<td>15</td>
</tr>
</tbody>
</table>

The information displayed in the DATA from pg_buffercache section of the report includes:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELATION</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>BUFFERS</td>
<td>The number of shared buffers used by the relation.</td>
</tr>
</tbody>
</table>

**Note:** In order to obtain the report for DATA from pg_buffercache, the pg_buffercache module must have been installed in the database. Perform the installation with the `CREATE EXTENSION` command.

For more information on the `CREATE EXTENSION` command please see the PostgreSQL Core documentation at:

[https://www.postgresql.org/docs/9.5/static/sql-createextension.html](https://www.postgresql.org/docs/9.5/static/sql-createextension.html)

**DATA from pg_stat_all_tables ordered by seq scan**

<table>
<thead>
<tr>
<th>SCHEMA</th>
<th>RELATION</th>
<th>IDX SCAN</th>
<th>IDX TUP READ</th>
<th>INS</th>
<th>UPD</th>
<th>DEL</th>
<th>SEQ SCAN</th>
<th>REL TUP READ</th>
</tr>
</thead>
<tbody>
<tr>
<td>pg_catalog</td>
<td>pg_class</td>
<td>546</td>
<td>319</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>16</td>
<td>7162</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_am</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_database</td>
<td>42</td>
<td>42</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>16</td>
</tr>
</tbody>
</table>
The information displayed in the DATA from pg_stat_all_tables ordered by seq scan section includes:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA</td>
<td>The name of the schema in which the table resides.</td>
</tr>
<tr>
<td>RELATION</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>SEQ SCAN</td>
<td>The number of sequential scans initiated on this table.</td>
</tr>
<tr>
<td>REL TUP READ</td>
<td>The number of tuples read in the table.</td>
</tr>
<tr>
<td>IDX SCAN</td>
<td>The number of index scans initiated on the table.</td>
</tr>
<tr>
<td>IDX TUP READ</td>
<td>The number of index tuples read.</td>
</tr>
<tr>
<td>INS</td>
<td>The number of rows inserted.</td>
</tr>
<tr>
<td>UPD</td>
<td>The number of rows updated.</td>
</tr>
<tr>
<td>DEL</td>
<td>The number of rows deleted.</td>
</tr>
</tbody>
</table>

### DATA from pg_stat_all_tables ordered by rel tup read

<table>
<thead>
<tr>
<th>SCHEMA</th>
<th>RELATION</th>
<th>SEQ SCAN</th>
<th>REL TUP READ</th>
</tr>
</thead>
<tbody>
<tr>
<td>pg_catalog</td>
<td>pg_class</td>
<td>16</td>
<td>7162</td>
</tr>
<tr>
<td>546</td>
<td>319</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_index</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>145</td>
<td>149</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_namespace</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>49</td>
<td>49</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_database</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>42</td>
<td>42</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_am</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sys</td>
<td>edb$session_wait_history</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_authid</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sys</td>
<td>edb$session_wait_history</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sys</td>
<td>edb$session_waits</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sys</td>
<td>edb$stat_all_indexes</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
The information displayed in the Data from pg_statio_all_tables ordered by rel tup read section includes:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA</td>
<td>The name of the schema in which the table resides.</td>
</tr>
<tr>
<td>RELATION</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>SEQ SCAN</td>
<td>The number of sequential scans performed on the table.</td>
</tr>
<tr>
<td>REL TUP READ</td>
<td>The number of tuples read from the table.</td>
</tr>
<tr>
<td>IDX SCAN</td>
<td>The number of index scans performed on the table.</td>
</tr>
<tr>
<td>IDX TUP READ</td>
<td>The number of index tuples read.</td>
</tr>
<tr>
<td>INS</td>
<td>The number of rows inserted.</td>
</tr>
<tr>
<td>UPD</td>
<td>The number of rows updated.</td>
</tr>
<tr>
<td>DEL</td>
<td>The number of rows deleted.</td>
</tr>
</tbody>
</table>

**Data from pg_statio_all_tables**

<table>
<thead>
<tr>
<th>SCHEMA</th>
<th>RELATION</th>
<th>HEAP READ</th>
<th>HEAP HIT</th>
<th>HEAP ICACHE</th>
<th>IDX READ</th>
<th>IDX HIT</th>
<th>IDX ICACHE</th>
<th>IDX READ</th>
<th>IDX HIT</th>
<th>IDX ICACHE</th>
<th>IDX READ</th>
<th>IDX HIT</th>
<th>IDX ICACHE</th>
<th>IDX READ</th>
<th>IDX HIT</th>
<th>IDX ICACHE</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>pgbench_accounts</td>
<td>92766</td>
<td>67215</td>
<td>288</td>
<td>59</td>
<td>32126</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_class</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sys</td>
<td>edb$stat_all_indexes</td>
<td>8</td>
<td>125</td>
<td>0</td>
<td>4</td>
<td>233</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sys</td>
<td>edb$statio_all_index</td>
<td>8</td>
<td>125</td>
<td>0</td>
<td>4</td>
<td>233</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sys</td>
<td>edb$stat_all_tables</td>
<td>6</td>
<td>91</td>
<td>0</td>
<td>2</td>
<td>174</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sys</td>
<td>edb$statio_all_table</td>
<td>6</td>
<td>91</td>
<td>0</td>
<td>2</td>
<td>174</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_namespace</td>
<td>3</td>
<td>72</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sys</td>
<td>edb$session_wait_his</td>
<td>1</td>
<td>24</td>
<td>0</td>
<td>4</td>
<td>47</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_opclass</td>
<td>3</td>
<td>13</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_trigger</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>1</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The information displayed in the Data from pg_statio_all_tables section includes:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA</td>
<td>The name of the schema in which the table resides.</td>
</tr>
<tr>
<td>RELATION</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>HEAP READ</td>
<td>The number of heap blocks read.</td>
</tr>
<tr>
<td>HEAP HIT</td>
<td>The number of heap blocks hit.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HEAP ICACHE HIT</td>
<td>The number of heap blocks in Infinite Cache.</td>
</tr>
<tr>
<td>IDX READ</td>
<td>The number of index blocks read.</td>
</tr>
<tr>
<td>IDX HIT</td>
<td>The number of index blocks hit.</td>
</tr>
<tr>
<td>IDX ICACHE HIT</td>
<td>The number of index blocks in Infinite Cache.</td>
</tr>
<tr>
<td>TOAST READ</td>
<td>The number of toast blocks read.</td>
</tr>
<tr>
<td>TOAST HIT</td>
<td>The number of toast blocks hit.</td>
</tr>
<tr>
<td>TOAST ICACHE HIT</td>
<td>The number of toast blocks in Infinite Cache.</td>
</tr>
<tr>
<td>TIDX READ</td>
<td>The number of toast index blocks read.</td>
</tr>
<tr>
<td>TIDX HIT</td>
<td>The number of toast index blocks hit.</td>
</tr>
<tr>
<td>TIDX ICACHE HIT</td>
<td>The number of toast index blocks in Infinite Cache.</td>
</tr>
</tbody>
</table>

The information displayed in the **DATA from pg_stat_all_indexes** section includes:

<table>
<thead>
<tr>
<th>SCHEMA</th>
<th>RELATION</th>
<th>INDEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>pg_catalog</td>
<td>pg_attribute</td>
<td>907</td>
</tr>
<tr>
<td>pg_attribute_relid_attnum_index</td>
<td>427</td>
<td>pg_class_rename_nsp_index</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_class</td>
<td>62</td>
</tr>
<tr>
<td>pg_attribute_relid_attnum_index</td>
<td>289</td>
<td>pg_class_oid_index</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_class</td>
<td>257</td>
</tr>
<tr>
<td>pg_attribute_relid_attnum_index</td>
<td>257</td>
<td>pg_class_oid_index</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_statistic</td>
<td>207</td>
</tr>
<tr>
<td>pg_statistic_relid_atinh_index</td>
<td>207</td>
<td>pg_class_oid_index</td>
</tr>
<tr>
<td>enterprisedb</td>
<td>pgbench_accounts</td>
<td>200</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_cast</td>
<td>255</td>
</tr>
<tr>
<td>pg_cast_source_target_index</td>
<td>200</td>
<td>pg_class_oid_index</td>
</tr>
<tr>
<td>100</td>
<td>pg_cast</td>
<td>112</td>
</tr>
<tr>
<td>pg_cast_source_target_index</td>
<td>112</td>
<td>pg_class_oid_index</td>
</tr>
<tr>
<td>101</td>
<td>pg_cast</td>
<td>110</td>
</tr>
<tr>
<td>pg_cast_source_target_index</td>
<td>101</td>
<td>pg_class_oid_index</td>
</tr>
</tbody>
</table>

The information displayed in the **DATA from pg_stat_all_indexes** section includes:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA</td>
<td>The name of the schema in which the index resides.</td>
</tr>
<tr>
<td>RELATION</td>
<td>The name of the table on which the index is defined.</td>
</tr>
<tr>
<td>INDEX</td>
<td>The name of the index.</td>
</tr>
<tr>
<td>IDX SCAN</td>
<td>The number of indexes scans initiated on this index.</td>
</tr>
<tr>
<td>IDX TUP READ</td>
<td>Number of index entries returned by scans on this index.</td>
</tr>
<tr>
<td>IDX TUP FETCH</td>
<td>Number of live table rows fetched by simple index scans using this index.</td>
</tr>
</tbody>
</table>

The information displayed in the **DATA from pg_statio_all_indexes** section includes:
<table>
<thead>
<tr>
<th>SCHEMA</th>
<th>RELATION</th>
<th>INDEX</th>
<th>IDX BLKS READ</th>
<th>IDX BLKS HIT</th>
<th>IDX BLKS ICACHE HIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>pg_catalog</td>
<td>pg_attribute</td>
<td></td>
<td>0</td>
<td>867</td>
<td>0</td>
</tr>
<tr>
<td>enterprisedb</td>
<td>pgbench_accounts</td>
<td>pgbench_accounts_pkey</td>
<td>0</td>
<td>778</td>
<td>0</td>
</tr>
<tr>
<td>pg_class</td>
<td>pg_class_oid_index</td>
<td></td>
<td>0</td>
<td>590</td>
<td>0</td>
</tr>
<tr>
<td>pg_class</td>
<td>pg_class_oid_index</td>
<td></td>
<td>0</td>
<td>527</td>
<td>0</td>
</tr>
<tr>
<td>pg_class</td>
<td>pg_statistic</td>
<td></td>
<td>0</td>
<td>441</td>
<td>0</td>
</tr>
<tr>
<td>sys</td>
<td>edb$stat_all_indexes</td>
<td>edb$stat_idx_pk</td>
<td>0</td>
<td>332</td>
<td>0</td>
</tr>
<tr>
<td>sys</td>
<td>edb$stat_all_indexes</td>
<td>edb$stat_idx_pk</td>
<td>1</td>
<td>332</td>
<td>0</td>
</tr>
<tr>
<td>pg_class</td>
<td>pg_proc</td>
<td></td>
<td>0</td>
<td>244</td>
<td>0</td>
</tr>
<tr>
<td>edb$stat_all_tables</td>
<td>edb$stat_tab_pk</td>
<td></td>
<td>0</td>
<td>241</td>
<td>0</td>
</tr>
<tr>
<td>edb$stat_all_tables</td>
<td>edb$stat_tab_pk</td>
<td></td>
<td>0</td>
<td>241</td>
<td>0</td>
</tr>
</tbody>
</table>

The information displayed in the **DATA from pg_statio_all_indexes** section includes:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA</td>
<td>The name of the schema in which the index resides.</td>
</tr>
<tr>
<td>RELATION</td>
<td>The name of the table on which the index is defined.</td>
</tr>
<tr>
<td>INDEX</td>
<td>The name of the index.</td>
</tr>
<tr>
<td>IDX BLKS READ</td>
<td>The number of index blocks read.</td>
</tr>
<tr>
<td>IDX BLKS HIT</td>
<td>The number of index blocks hit.</td>
</tr>
<tr>
<td>IDX BLKS ICACHE HIT</td>
<td>The number of index blocks in Infinite Cache that were hit.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System Wait Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WAIT NAME</strong></td>
</tr>
<tr>
<td>-----------------------</td>
</tr>
<tr>
<td>query plan</td>
</tr>
<tr>
<td>db file read</td>
</tr>
</tbody>
</table>

The information displayed in the **System Wait Information** section includes:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAIT_NAME</td>
<td>The name of the wait.</td>
</tr>
<tr>
<td>COUNT</td>
<td>The number of times that the wait event occurred.</td>
</tr>
<tr>
<td>WAIT_TIME</td>
<td>The length of the wait time in milliseconds.</td>
</tr>
<tr>
<td>% WAIT</td>
<td>The percentage of the total wait time used by this wait for this session.</td>
</tr>
</tbody>
</table>

Database Parameters from postgresql.conf
### Database Parameters from postgresql.conf

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>SETTING</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTEXT MINVAL MAXVAL</td>
<td></td>
</tr>
<tr>
<td>allow_system_table_mods</td>
<td>off</td>
</tr>
<tr>
<td>postmaster</td>
<td></td>
</tr>
<tr>
<td>application_name</td>
<td>psql</td>
</tr>
<tr>
<td>user</td>
<td></td>
</tr>
<tr>
<td>archive_command</td>
<td>(disabled)</td>
</tr>
<tr>
<td>sighup</td>
<td></td>
</tr>
<tr>
<td>archive_mode</td>
<td>off</td>
</tr>
<tr>
<td>postmaster</td>
<td></td>
</tr>
<tr>
<td>archive_timeout</td>
<td>0</td>
</tr>
<tr>
<td>sighup</td>
<td>0</td>
</tr>
<tr>
<td>array_nulls</td>
<td>on</td>
</tr>
<tr>
<td>user</td>
<td></td>
</tr>
<tr>
<td>authentication_timeout</td>
<td>60</td>
</tr>
<tr>
<td>sighup</td>
<td>1</td>
</tr>
<tr>
<td>autovacuum</td>
<td>on</td>
</tr>
<tr>
<td>sighup</td>
<td></td>
</tr>
<tr>
<td>autovacuum_analyze_scale_factor</td>
<td>0.1</td>
</tr>
<tr>
<td>sighup</td>
<td>0</td>
</tr>
<tr>
<td>autovacuum_analyze_threshold</td>
<td>50</td>
</tr>
<tr>
<td>sighup</td>
<td>0</td>
</tr>
<tr>
<td>autovacuum_freeze_max_age</td>
<td>2000000000</td>
</tr>
<tr>
<td>postmaster</td>
<td></td>
</tr>
<tr>
<td>1000000000</td>
<td></td>
</tr>
<tr>
<td>autovacuum_max_workers</td>
<td>3</td>
</tr>
<tr>
<td>postmaster</td>
<td></td>
</tr>
<tr>
<td>8388607</td>
<td></td>
</tr>
<tr>
<td>autovacuum_naptime</td>
<td>60</td>
</tr>
<tr>
<td>sighup</td>
<td>1</td>
</tr>
<tr>
<td>autovacuum_vacuum_cost_delay</td>
<td>20</td>
</tr>
</tbody>
</table>

The information displayed in the **Database Parameters from postgresql.conf** section includes:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARAMETER</td>
<td>The name of the parameter.</td>
</tr>
<tr>
<td>SETTING</td>
<td>The current value assigned to the parameter.</td>
</tr>
<tr>
<td>CONTEXT</td>
<td>The context required to set the parameter value.</td>
</tr>
<tr>
<td>MINVAL</td>
<td>The minimum value allowed for the parameter.</td>
</tr>
<tr>
<td>MAXVAL</td>
<td>The maximum value allowed for the parameter.</td>
</tr>
</tbody>
</table>

### 5.3.2 stat_db_rpt()

The signature is:

```
stat_db_rpt(beginning_id, ending_id)
```

**Parameters**

- `beginning_id`
beginning_id is an integer value that represents the beginning session identifier.

ending_id

ending_id is an integer value that represents the ending session identifier.

The following example demonstrates the stat_db_rpt() function:

```sql
SELECT * FROM stat_db_rpt(9, 10);
```

<table>
<thead>
<tr>
<th>DATABASE</th>
<th>NUMBACKENDS</th>
<th>XACT COMMIT</th>
<th>XACT ROLLBACK</th>
<th>BLKS READ</th>
<th>BLKS HIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>edb</td>
<td>1</td>
<td>21</td>
<td>0</td>
<td>92928</td>
<td>101217</td>
</tr>
</tbody>
</table>

The information displayed in the DATA from pg_stat_database section of the report includes:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATABASE</td>
<td>The name of the database.</td>
</tr>
<tr>
<td>NUMBACKENDS</td>
<td>Number of backends currently connected to this database. This is the only column in this view that returns a value reflecting current state; all other columns return the accumulated values since the last reset.</td>
</tr>
<tr>
<td>XACT COMMIT</td>
<td>The number of transactions in this database that have been committed.</td>
</tr>
<tr>
<td>XACT ROLLBACK</td>
<td>The number of transactions in this database that have been rolled back.</td>
</tr>
<tr>
<td>BLKS READ</td>
<td>The number of blocks read.</td>
</tr>
<tr>
<td>BLKS HIT</td>
<td>The number of blocks hit.</td>
</tr>
<tr>
<td>BLKS ICACHE HIT</td>
<td>The number of blocks in Infinite Cache that were hit.</td>
</tr>
<tr>
<td>HIT RATIO</td>
<td>The percentage of times that a block was found in the shared buffer cache.</td>
</tr>
<tr>
<td>ICACHE HIT RATIO</td>
<td>The percentage of times that a block was found in Infinite Cache.</td>
</tr>
</tbody>
</table>

5.3.3 stat_tables_rpt()

The signature is:

```sql
function_name(beginning_id, ending_id, top_n, scope)
```

Parameters

beginning_id
beginning_id is an integer value that represents the beginning session identifier.

ending_id

ending_id is an integer value that represents the ending session identifier.

top_n

top_n represents the number of rows to return

scope

scope determines which tables the function returns statistics about. Specify SYS, USER or ALL:

- SYS indicates that the function should return information about system defined tables. A table is considered a system table if it is stored in one of the following schemas: pg_catalog, information_schema, sys, or dbo.

- USER indicates that the function should return information about user-defined tables.

- ALL specifies that the function should return information about all tables.

The stat_tables_rpt() function returns a two-part report. The first portion of the report contains:

```
SELECT * FROM stat_tables_rpt(18, 19, 10, 'ALL');
```

<table>
<thead>
<tr>
<th>SCHEMA</th>
<th>RELATION</th>
<th>SEQ SCAN</th>
<th>REL TUP READ</th>
<th>IDX SCAN</th>
<th>IDX TUP READ</th>
<th>INS</th>
<th>UPD</th>
<th>DEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>pg_catalog</td>
<td>pg_class</td>
<td>2952</td>
<td>78</td>
<td>65</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_index</td>
<td>448</td>
<td>23</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_namespace</td>
<td>76</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_database</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_authid</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sys</td>
<td>$snap</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
The information displayed in the DATA from pg_stat_all_tables ordered by seq scan section includes:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA</td>
<td>The name of the schema in which the table resides.</td>
</tr>
<tr>
<td>RELATION</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>SEQ SCAN</td>
<td>The number of sequential scans on the table.</td>
</tr>
<tr>
<td>REL TUP READ</td>
<td>The number of tuples read from the table.</td>
</tr>
<tr>
<td>IDX SCAN</td>
<td>The number of index scans performed on the table.</td>
</tr>
<tr>
<td>IDX TUP READ</td>
<td>The number of index tuples read from the table.</td>
</tr>
<tr>
<td>INS</td>
<td>The number of rows inserted.</td>
</tr>
<tr>
<td>UPD</td>
<td>The number of rows updated.</td>
</tr>
<tr>
<td>DEL</td>
<td>The number of rows deleted.</td>
</tr>
</tbody>
</table>

The second portion of the report contains:

```
DATA from pg_stat_all_tables ordered by rel tup read

<table>
<thead>
<tr>
<th>SCHEMA</th>
<th>RELATION</th>
<th>SEQ SCAN</th>
<th>REL TUP READ</th>
<th>IDX SCAN</th>
<th>IDX TUP READ</th>
<th>INS</th>
<th>UPD</th>
<th>DEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>pg_catalog</td>
<td>pg_class</td>
<td>8</td>
<td>2952</td>
<td>78</td>
<td>65</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_index</td>
<td>4</td>
<td>448</td>
<td>23</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_namespace</td>
<td>4</td>
<td>76</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sys</td>
<td>edb$snap</td>
<td>1</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_database</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_authid</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>public</td>
<td>accounts</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>public</td>
<td>branches</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sys</td>
<td>edb$session_wait_history</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>25</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sys</td>
<td>edb$session_waits</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
(29 rows)
```
The information displayed in the `DATA` from `pg_stat_all_tables` ordered by `rel tup read` section includes:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA</td>
<td>The name of the schema in which the table resides.</td>
</tr>
<tr>
<td>RELATION</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>SEQ SCAN</td>
<td>The number of sequential scans performed on the table.</td>
</tr>
<tr>
<td>REL TUP READ</td>
<td>The number of tuples read from the table.</td>
</tr>
<tr>
<td>IDX SCAN</td>
<td>The number of index scans performed on the table.</td>
</tr>
<tr>
<td>IDX TUP READ</td>
<td>The number of live rows fetched by index scans.</td>
</tr>
<tr>
<td>INS</td>
<td>The number of rows inserted.</td>
</tr>
<tr>
<td>UPD</td>
<td>The number of rows updated.</td>
</tr>
<tr>
<td>DEL</td>
<td>The number of rows deleted.</td>
</tr>
</tbody>
</table>

### 5.3.4 statio_tables_rpt()

The signature is:

```
statio_tables_rpt(beginning_id, ending_id, top_n, scope)
```

**Parameters**

- **beginning_id**
  
  `beginning_id` is an integer value that represents the beginning session identifier.

- **ending_id**
  
  `ending_id` is an integer value that represents the ending session identifier.

- **top_n**
  
  `top_n` represents the number of rows to return

- **scope**
  
  `scope` determines which tables the function returns statistics about. Specify `SYS`, `USER` or `ALL`:
  
  - `SYS` indicates that the function should return information about system defined tables. A table is considered a system table if it is stored in one of
the following schemas: pg_catalog, information_schema, sys, or dbo.

- **USER** indicates that the function should return information about user-defined tables.

- **ALL** specifies that the function should return information about all tables.

The `statio_tables_rpt()` function returns a report that contains:

```sql
edb=# SELECT * FROM statio_tables_rpt(9, 10, 10, 'SYS');
```

<table>
<thead>
<tr>
<th>SCHEMA</th>
<th>RELATION</th>
<th>HEAP READ</th>
<th>HEAP HIT</th>
<th>HEAP ICACHE HIT</th>
<th>IDX READ</th>
<th>IDX HIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>pgbench_accounts</td>
<td>92766</td>
<td>67215</td>
<td>288</td>
<td>59</td>
<td>32126</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_class</td>
<td>0</td>
<td>296</td>
<td>0</td>
<td>3</td>
<td>16</td>
</tr>
<tr>
<td>sys</td>
<td>edb$stat_all_indexes</td>
<td>8</td>
<td>125</td>
<td>0</td>
<td>4</td>
<td>233</td>
</tr>
<tr>
<td>sys</td>
<td>edb$statio_all_index</td>
<td>8</td>
<td>125</td>
<td>0</td>
<td>4</td>
<td>233</td>
</tr>
<tr>
<td>sys</td>
<td>edb$stat_all_tables</td>
<td>6</td>
<td>91</td>
<td>0</td>
<td>2</td>
<td>174</td>
</tr>
<tr>
<td>sys</td>
<td>edb$statio_all_table</td>
<td>6</td>
<td>91</td>
<td>0</td>
<td>2</td>
<td>174</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_namespace</td>
<td>3</td>
<td>72</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sys</td>
<td>edb$session_wait_his</td>
<td>1</td>
<td>24</td>
<td>0</td>
<td>4</td>
<td>47</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_opclass</td>
<td>3</td>
<td>13</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_trigger</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>1</td>
<td>15</td>
</tr>
</tbody>
</table>
```

(16 rows)

The information displayed in the `Data from pg_statio_all_tables` section includes:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA</td>
<td>The name of the schema in which the relation resides.</td>
</tr>
<tr>
<td>RELATION</td>
<td>The name of the relation.</td>
</tr>
<tr>
<td>HEAP READ</td>
<td>The number of heap blocks read.</td>
</tr>
<tr>
<td>HEAP HIT</td>
<td>The number of heap blocks hit.</td>
</tr>
</tbody>
</table>
### 5.3.5 stat_indexes_rpt()

The signature is:

\[
\text{stat_indexes_rpt}(\text{beginning\_id}, \text{ending\_id}, \text{top\_n}, \text{scope})
\]

**Parameters**

**beginning\_id**

- \text{beginning\_id} is an integer value that represents the beginning session identifier.

**ending\_id**

- \text{ending\_id} is an integer value that represents the ending session identifier.

**top\_n**

- \text{top\_n} represents the number of rows to return

**scope**

- \text{scope} determines which tables the function returns statistics about. Specify \text{SYS}, \text{USER} or \text{ALL}:
  
  - \text{SYS} indicates that the function should return information about system defined tables. A table is considered a system table if it is stored in one of the following schemas: \text{pg\_catalog}, \text{information\_schema}, \text{sys}, or \text{dbo}.
- **USER** indicates that the function should return information about user-defined tables.

- **ALL** specifies that the function should return information about all tables.

The `stat_indexes_rpt()` function returns a report that contains:

```sql
edb=# SELECT * FROM stat_indexes_rpt(9, 10, 10, 'ALL');

<table>
<thead>
<tr>
<th>SCHEMA</th>
<th>RELATION</th>
<th>INDEX</th>
<th>IDX SCAN</th>
<th>IDX TUP READ</th>
<th>IDX TUP FETCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>pg_catalog</td>
<td>pg_cast</td>
<td>pg_cast_source_target_index</td>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_class</td>
<td>pg_class_oid_index</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_trigger</td>
<td>pg_trigger_tgrelid_tgname_index</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_attribute</td>
<td>pg_attribute_relid_attnum_index</td>
<td>7</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_statistic</td>
<td>pg_statistic_relid_att_index</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_database</td>
<td>pg_database_oid_index</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_proc</td>
<td>pg_proc_oid_index</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_operator</td>
<td>pg_operator_oprname_l_r_n_index</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_type</td>
<td>pg_type_typname_nsp_index</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>pg_catalog</td>
<td>pg_amop</td>
<td>pg_amop_opr_fam_index</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
```

(14 rows)

The information displayed in the **DATA from pg_stat_all_indexes** section includes:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA</td>
<td>The name of the schema in which the relation resides.</td>
</tr>
<tr>
<td>RELATION</td>
<td>The name of the relation.</td>
</tr>
<tr>
<td>INDEX</td>
<td>The name of the index.</td>
</tr>
<tr>
<td>IDX SCAN</td>
<td>The number of indexes scanned.</td>
</tr>
<tr>
<td>IDX TUP READ</td>
<td>The number of index tuples read.</td>
</tr>
<tr>
<td>IDX TUP FETCH</td>
<td>The number of index tuples fetched.</td>
</tr>
</tbody>
</table>
5.3.6 statio_indexes_rpt()

The signature is:

```
statio_indexes_rpt(beginning_id, ending_id, top_n, scope)
```

Parameters

beginning_id

`beginning_id` is an integer value that represents the beginning session identifier.

ending_id

`ending_id` is an integer value that represents the ending session identifier.

top_n

top_n represents the number of rows to return.

scope

`scope` determines which tables the function returns statistics about. Specify SYS, USER or ALL:

- **SYS** indicates that the function should return information about system defined tables. A table is considered a system table if it is stored in one of the following schemas: `pg_catalog`, `information_schema`, `sys`, or `dbo`.

- **USER** indicates that the function should return information about user-defined tables.

- **ALL** specifies that the function should return information about all tables.

The `statio_indexes_rpt()` function returns a report that contains:

```
edb=# SELECT * FROM statio_indexes_rpt(9, 10, 10, 'SYS');

statio_indexes_rpt
-------------------------------
DATA from pg_statio_all_indexes

<table>
<thead>
<tr>
<th>SCHEMA</th>
<th>RELATION</th>
<th>INDEX</th>
<th>IDX BLKS READ</th>
<th>IDX BLKS HIT</th>
<th>IDX BLKS ICACHE HIT</th>
</tr>
</thead>
</table>
```
The information displayed in the DATA from `pg_statio_all_indexes` report includes:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA</td>
<td>The name of the schema in which the relation resides.</td>
</tr>
<tr>
<td>RELATION</td>
<td>The name of the table on which the index is defined.</td>
</tr>
<tr>
<td>INDEX</td>
<td>The name of the index.</td>
</tr>
<tr>
<td>IDX BLKS READ</td>
<td>The number of index blocks read.</td>
</tr>
<tr>
<td>IDX BLKS HIT</td>
<td>The number of index blocks hit.</td>
</tr>
<tr>
<td>IDX BLKS ICACHE HIT</td>
<td>The number of index blocks in Infinite Cache that were hit.</td>
</tr>
</tbody>
</table>
5.4 Performance Tuning Recommendations

To use DRITA reports for performance tuning, review the top five events in a given report, looking for any event that takes a disproportionately large percentage of resources. In a streamlined system, user I/O will probably make up the largest number of waits. Waits should be evaluated in the context of CPU usage and total time; an event may not be significant if it takes 2 minutes out of a total measurement interval of 2 hours, if the rest of the time is consumed by CPU time. The component of response time (CPU "work" time or other "wait" time) that consumes the highest percentage of overall time should be evaluated.

When evaluating events, watch for:

<table>
<thead>
<tr>
<th>Event type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checkpoint waits</td>
<td>Checkpoint waits may indicate that checkpoint parameters need to be adjusted, (checkpoint_segments and checkpoint_timeout).</td>
</tr>
<tr>
<td>WAL-related waits</td>
<td>WAL-related waits may indicate wal_buffers are under-sized.</td>
</tr>
<tr>
<td>SQL Parse waits</td>
<td>If the number of waits is high, try to use prepared statements.</td>
</tr>
<tr>
<td>db file random reads</td>
<td>If high, check that appropriate indexes and statistics exist.</td>
</tr>
<tr>
<td>db file random writes</td>
<td>If high, may need to decrease bgwriter_delay.</td>
</tr>
<tr>
<td>btree random lock acquires</td>
<td>May indicate indexes are being rebuilt. Schedule index builds during less active time.</td>
</tr>
</tbody>
</table>

Performance reviews should also include careful scrutiny of the hardware, the operating system, the network and the application SQL statements.
### 5.5 Event Descriptions

The following table lists the basic wait events that are displayed by DRITA.

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add in shmem lock acquire</td>
<td>Obsolete/unused</td>
</tr>
<tr>
<td>bgwriter communication lock acquire</td>
<td>The bgwriter (background writer) process has waited for the short-term lock that synchronizes messages between the bgwriter and a backend process.</td>
</tr>
<tr>
<td>btree vacuum lock acquire</td>
<td>The server has waited for the short-term lock that synchronizes access to the next available vacuum cycle ID.</td>
</tr>
<tr>
<td>buffer free list lock acquire</td>
<td>The server has waited for the short-term lock that synchronizes access to the list of free buffers (in shared memory).</td>
</tr>
<tr>
<td>checkpoint lock acquire:</td>
<td>A server process has waited for the short-term lock that prevents simultaneous checkpoints.</td>
</tr>
<tr>
<td>checkpoint start lock acquire</td>
<td>The server has waited for the short-term lock that synchronizes access to the bgwriter checkpoint schedule.</td>
</tr>
<tr>
<td>clog control lock acquire</td>
<td>The server has waited for the short-term lock that synchronizes access to the commit log.</td>
</tr>
<tr>
<td>control file lock acquire</td>
<td>The server has waited for the short-term lock that synchronizes write access to the control file (this should usually be a low number).</td>
</tr>
<tr>
<td>db file extend</td>
<td>A server process has waited for the operating system while adding a new page to the end of a file.</td>
</tr>
<tr>
<td>db file read</td>
<td>A server process has waited for the completion of a read (from disk).</td>
</tr>
<tr>
<td>db file write</td>
<td>A server process has waited for the completion of a write (to disk).</td>
</tr>
<tr>
<td>db file sync</td>
<td>A server process has waited for the operating system to flush all changes to disk.</td>
</tr>
<tr>
<td>first buf mapping lock acquire</td>
<td>The server has waited for a short-term lock that synchronizes access to the shared-buffer mapping table.</td>
</tr>
<tr>
<td>freespace lock acquire</td>
<td>The server has waited for the short-term lock that synchronizes access to the freespace map.</td>
</tr>
<tr>
<td>Infinite Cache read</td>
<td>The server has waited for an Infinite Cache read request.</td>
</tr>
<tr>
<td>Infinite Cache write</td>
<td>The server has waited for an Infinite Cache write request.</td>
</tr>
<tr>
<td>lwlock acquire</td>
<td>The server has waited for a short-term lock that has not been described elsewhere in this section.</td>
</tr>
<tr>
<td>multi xact gen lock acquire</td>
<td>The server has waited for the short-term lock that synchronizes access to the next available multi-transaction ID (when a SELECT...FOR SHARE statement executes).</td>
</tr>
<tr>
<td>multi xact member lock acquire</td>
<td>The server has waited for the short-term lock that synchronizes access to the multi-transaction member file (when a SELECT...FOR SHARE statement executes).</td>
</tr>
<tr>
<td>multi xact offset lock acquire</td>
<td>The server has waited for the short-term lock that synchronizes access to the multi-transaction offset file (when a SELECT...FOR SHARE statement executes).</td>
</tr>
<tr>
<td>oid gen lock acquire</td>
<td>The server has waited for the short-term lock that synchronizes access to the next available OID (object ID).</td>
</tr>
<tr>
<td>query plan</td>
<td>The server has computed the execution plan for a SQL statement.</td>
</tr>
<tr>
<td>rel cache init lock acquire</td>
<td>The server has waited for the short-term lock that prevents simultaneous relation-cache loads/unloads.</td>
</tr>
<tr>
<td>Event Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>shmem index lock acquire</td>
<td>The server has waited for the short-term lock that synchronizes access to the shared-memory map.</td>
</tr>
<tr>
<td>sinval lock acquire</td>
<td>The server has waited for the short-term lock that synchronizes access to the cache invalidation state.</td>
</tr>
<tr>
<td>sql parse</td>
<td>The server has parsed a SQL statement.</td>
</tr>
<tr>
<td>subtrans control lock acquire</td>
<td>The server has waited for the short-term lock that synchronizes access to the subtransaction log.</td>
</tr>
<tr>
<td>tablespace create lock acquire</td>
<td>The server has waited for the short-term lock that prevents simultaneous CREATE TABLESPACE or DROP TABLESPACE commands.</td>
</tr>
<tr>
<td>two phase state lock acquire</td>
<td>The server has waited for the short-term lock that synchronizes access to the list of prepared transactions.</td>
</tr>
<tr>
<td>wal insert lock acquire</td>
<td>The server has waited for the short-term lock that synchronizes write access to the write-ahead log. A high number may indicate that WAL buffers are sized too small.</td>
</tr>
<tr>
<td>wal write lock acquire</td>
<td>The server has waited for the short-term lock that synchronizes write-ahead log flushes.</td>
</tr>
<tr>
<td>wal file sync</td>
<td>The server has waited for the write-ahead log to sync to disk (related to the wal_sync_method parameter which, by default, is 'fsync' - better performance can be gained by changing this parameter to open_sync).</td>
</tr>
<tr>
<td>wal flush</td>
<td>The server has waited for the write-ahead log to flush to disk.</td>
</tr>
<tr>
<td>wal write</td>
<td>The server has waited for a write to the write-ahead log buffer (expect this value to be high).</td>
</tr>
<tr>
<td>xid gen lock acquire</td>
<td>The server has waited for the short-term lock that synchronizes access to the next available transaction ID.</td>
</tr>
</tbody>
</table>

When wait events occur for lightweight locks, they are displayed by DRITA as well. A lightweight lock is used to protect a particular data structure in shared memory.

Certain wait events can be due to the server process waiting for one of a group of related lightweight locks, which is referred to as a lightweight lock tranche. Individual lightweight lock tranches are not displayed by DRITA, but their summation is displayed by a single event named other lwlock acquire.

For a list and description of lightweight locks displayed by DRITA, please see Section 27.2, The Statistics Collector in the PostgreSQL core documentation available at:

https://www.postgresql.org/docs/9.6/static/monitoring-stats.html

Under Section 27.2.2. Viewing Statistics, the lightweight locks are listed in Table 27-4 wait_event Description where the Wait Event Type column designates LWLockNamed.

Note that the table entries where Wait Event Type designates LWLockTranche are the lightweight lock tranches.

The following example displays lightweight locks ProcArrayLock, CLogControlLock, WALBufMappingLock, and XidGenLock.
postgres=# select * from sys_rpt(40,70,20);

<table>
<thead>
<tr>
<th>WAIT NAME</th>
<th>COUNT</th>
<th>WAIT TIME</th>
<th>% WAIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>wal flush</td>
<td>56107</td>
<td>44.456494</td>
<td>47.65</td>
</tr>
<tr>
<td>db file read</td>
<td>66123</td>
<td>19.543968</td>
<td>20.95</td>
</tr>
<tr>
<td>wal write</td>
<td>32886</td>
<td>12.780866</td>
<td>13.70</td>
</tr>
<tr>
<td>wal file sync</td>
<td>32933</td>
<td>11.792972</td>
<td>12.64</td>
</tr>
<tr>
<td>query plan</td>
<td>223576</td>
<td>4.539186</td>
<td>4.87</td>
</tr>
<tr>
<td>db file extend</td>
<td>2339</td>
<td>0.087038</td>
<td>0.09</td>
</tr>
<tr>
<td>other lwlock acquire</td>
<td>402</td>
<td>0.066591</td>
<td>0.07</td>
</tr>
<tr>
<td>ProcArrayLock</td>
<td>135</td>
<td>0.012942</td>
<td>0.01</td>
</tr>
<tr>
<td>CLogControlLock</td>
<td>212</td>
<td>0.010333</td>
<td>0.01</td>
</tr>
<tr>
<td>WALBufMappingLock</td>
<td>47</td>
<td>0.006068</td>
<td>0.01</td>
</tr>
<tr>
<td>XidGenLock</td>
<td>53</td>
<td>0.005296</td>
<td>0.01</td>
</tr>
</tbody>
</table>

(13 rows)

DRITA also displays wait events that occur that are related to certain Advanced Server product features.

These Advanced Server feature specific wait events and the other lwlock acquire event are listed in the following table.

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICacheLock</td>
<td>The server has waited for access related to Infinite Cache.</td>
</tr>
<tr>
<td>BulkLoadLock</td>
<td>The server has waited for access related to EDB*Loader.</td>
</tr>
<tr>
<td>EDBObjectManagerLock</td>
<td>The server has waited for access related to EDB Resource Manager.</td>
</tr>
<tr>
<td>other lwlock acquire</td>
<td>Summation of waits for lightweight lock tranches.</td>
</tr>
</tbody>
</table>
6 Acknowledgements

The PostgreSQL 8.3, 8.4, 9.0, 9.1, 9.2, 9.3 and 9.4 Documentation provided the baseline for the portions of this guide that are common to PostgreSQL, and is hereby acknowledged:

Portions of this EnterpriseDB Software and Documentation may utilize the following copyrighted material, the use of which is hereby acknowledged.

PostgreSQL Documentation, Database Management System

PostgreSQL is Copyright © 1996-2016 by the PostgreSQL Global Development Group and is distributed under the terms of the license of the University of California below.

Postgres95 is Copyright © 1994-5 by the Regents of the University of California.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without a written agreement is hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copies.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS-IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.