



EDB

EDB Wait States

Release 1.0

EnterpriseDB Corporation

Jul 20, 2023

CONTENTS

1	Installing EDB Wait States	3
1.1	On Linux	3
1.2	On Windows	3
1.3	Uninstalling	4
2	Using EDB Wait States	5
2.1	Exploring data with the interface	5
2.1.1	edb_wait_states_data	6
2.1.2	Parameters	6
2.1.3	Example	7
2.1.4	edb_wait_states_queries	8
2.1.5	Parameters	8
2.1.6	Example	8
2.1.7	edb_wait_states_sessions	9
2.1.8	Parameters	9
2.1.9	Example	9
2.1.10	edb_wait_states_samples	10
2.1.11	Parameters	10
2.1.12	Example	11
2.1.13	edb_wait_states_purge	11
2.1.14	Example	12
2.2	Deactivating	12

EDB Wait States is a tool for analyzing performance and tuning by allowing the collection and querying of wait event data. Wait events, introduced in PostgreSQL 9.6, are recorded alongside other session activity and provide a snapshot of whether a session is waiting for I/O, CPU, IPC, locks, or timeouts. Snapshots of this information are gathered by the EDB Wait States background worker (BGW) at regular intervals.

The EDB Wait States interface allows you to control when and for how long the wait events are sampled and to extract the gathered samples in `edb_wait_states_data` for further analysis. By gathering this data over time, you can discover optimization opportunities and gain insight into what resources sessions are waiting on when performance is lower than expected.

INSTALLING EDB WAIT STATES

1.1 On Linux

To install EDB Wait States on a Linux system:

1. Install EDB Wait States with the `edb-asxx-server-edb_wait_states` RPM package, where `xx` is the EDB Postgres Advanced Server version number.
2. To launch the worker, register it in the `postgresql.conf` file using the `shared_preload_libraries` parameter. For example:

```
shared_preload_libraries = $libdir/edb_wait_states
```

1. Restart the database server. After a successful restart, the background worker begins collecting data.
2. To review the data, create the following extension:

```
CREATE EXTENSION edb_wait_states;
```

1. To terminate the EDB Wait States worker, remove `$libdir/edb_wait_states` from the `shared_preload_libraries` parameter, and restart the database server.

1.2 On Windows

To install EDB Wait States on a Windows system:

1. To install the EDB Wait States module with the EDB Modules installer, invoke the StackBuilder Plus utility. Follow the onscreen instructions to finish installing the EDB Modules .
2. To register the worker, modify the `postgresql.conf` file to include the EDB Wait States library in the `shared_preload_libraries` configuration parameter. The parameter value must include:

```
shared_preload_libraries = $libdir/edb_wait_states.dll
```

The EDB Wait States installation places the `edb_wait_states.dll` library file in the following path:

```
C:\Program Files\edb\as14\lib\
```

1. Restart the database server to make the changes take effect. After a successful restart, the background worker starts and begins collecting the data.
2. To view the data, create the following extension:

```
CREATE EXTENSION edb_wait_states;
```

The installer places the `edb_wait_states.control` file in the following path:

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

1.3 Uninstalling

To uninstall EDB Wait States:

1. Use the `DROP EXTENSION` command to drop the `edb_wait_states` extension.
2. Modify the `postgresql.conf` file, removing `$libdir/edb_wait_states.dll` from the `shared_preload_libraries` parameter.
3. To apply your changes, restart the database server.

USING EDB WAIT STATES

When EDB Wait States is registered as one of the shared preload libraries, it probes each of the running sessions at regular intervals. For every session, it collects information such as:

- The database to which it's connected.
- The logged-in user of the session.
- The query running in that session.
- The wait events on which it's waiting.

This information is saved in a set of files in a user-configurable path and directory folder given by the `edb_wait_states.directory` parameter to add to the `postgresql.conf` file. The path must be a full, absolute path, not a relative path.

2.1 Exploring data with the interface

Each of the functions in the EDB Wait States interface has common input and output parameters. Those parameters are:

Parameter(s)	Input or output	Description
<i>start_ts</i> and <i>end_ts</i>	Input	Together these specify the time interval and the data to read. If you specify only <i>start_ts</i> , the data starting from <i>start_ts</i> is output. If you specify only <i>end_ts</i> , data up to <i>end_ts</i> is output. If you don't specify either, all the data is output.
<i>query_id</i>	Output	Identifies a normalized query. It's internal hash code computed from the query.
<i>session_id</i>	Output	Identifies a session.
<i>ref_start_ts</i> and <i>ref_end_ts</i>	Output	Provides the timestamps of a file containing a particular data point. A data point might be a wait event sample record, a query record, or a session record.

The following examples use a scenario where three queries are executed simultaneously on four different sessions connected to different databases using different users. Those three queries are:

```
SELECT schemaname FROM pg_tables, pg_sleep(15) WHERE schemaname <>
pg_catalog; / *ranon2sessions* /
SELECT tablename FROM pg_tables, pg_sleep(10) WHERE schemaname <>
pg_catalog;
```

(continues on next page)

(continued from previous page)

```
SELECT tablename, schemaname FROM pg_tables, pg_sleep(10) WHERE schemaname
<> pg_catalog;
```

2.1.1 edb_wait_states_data

Use this function to read the data collected by the BGW:

```
edb_wait_states_data(
  IN start_ts timestamptz default -infinity::timestamptz,
  IN end_ts timestamptz default infinity::timestamptz,
  OUT session_id int4,
  OUT <dbname> text,
  OUT <username> text,
  OUT <query> text,
  OUT <query_start_time> timestamptz,
  OUT <sample_time> timestamptz,
  OUT <wait_event_type> text,
  OUT <wait_event> text
)
```

You can use this function to find out the following:

- The queries running in the given duration (defined by `start_ts` and `end_ts`) in all the sessions, and the wait events, if any, they were waiting on. For example:

```
SELECT query, session_id, wait_event_type, wait_event
FROM edb_wait_states_data(start_ts, end_ts);
```

- The progress of a session within a given duration, that is, the queries run in a session (`session_id = 100000`) and the wait events the queries waited on. For example:

```
SELECT query, wait_event_type, wait_event
FROM edb_wait_states_data(start_ts, end_ts)
WHERE session_id = 100000;
```

- The duration for which the samples are available. For example:

```
SELECT min(sample_time), max(sample_time)
FROM edb_wait_states_data();
```

2.1.2 Parameters

In addition to the common parameters described previously, each row of the output gives the following:

`dbname`

The session's database.

`username`

The session's logged-in user.

`query`

The query running in the session.

query_start_time

The time when the query started.

sample_time

The time when wait event data was collected.

wait_event_type

The type of wait event the session (backend) is waiting on.

wait_event

The wait event the session (backend) is waiting on.

2.1.3 Example

The following is a sample output from the `edb_wait_states_data()` function:

```
edb=# SELECT * FROM edb_wait_states_data();
__OUTPUT__
- [ RECORD 1 ]-----+-----
session_id      | 4398
dbname          | edb
username        | enterprisedb
query           | SELECT schemaname FROM pg_tables, pg_sleep($1) WHERE schemaname <> $2
query_start_time | 17-AUG-18 11:56:05.271962 -04:00
sample_time     | 17-AUG-18 11:56:19.700236 -04:00
wait_event_type  | Timeout
wait_event       | PgSleep
- [ RECORD 2 ]-----+-----
session_id      | 4398
dbname          | edb
username        | enterprisedb
query           | SELECT schemaname FROM pg_tables, pg_sleep($1) WHERE schemaname <> $2
query_start_time | 17-AUG-18 11:56:05.271962 -04:00
sample_time     | 17-AUG-18 11:56:18.699938 -04:00
wait_event_type  | Timeout
wait_event       | PgSleep
- [ RECORD 3 ]-----+-----
session_id      | 4398
dbname          | edb
username        | enterprisedb
query           | SELECT schemaname FROM pg_tables, pg_sleep($1) WHERE schemaname <> $2
query_start_time | 17-AUG-18 11:56:05.271962 -04:00
sample_time     | 17-AUG-18 11:56:17.700253 -04:00
wait_event_type  | Timeout
wait_event       | PgSleep
                  .
                  .
                  .
```

2.1.4 edb_wait_states_queries

This function gives information about the queries sampled by the BGW. For example:

```
edb_wait_states_queries(
  IN start_ts timestamptz default -infinity::timestamptz,
  IN end_ts timestamptz default infinity::timestamptz,
  OUT query_id int8,
  OUT <query> text,
  OUT ref_start_ts timestamptz
  OUT ref_end_ts timestamptz
)
```

A new queries file is created periodically. Multiple query files can be generated corresponding to specific intervals.

This function returns all the queries in query files that overlap with the given time interval. A query gives all the queries in query files that contained queries sampled between `start_ts` and `end_ts`:

```
SELECT query FROM edb_wait_states_queries(start_ts, end_ts);
```

In other words, the function can output queries that didn't run in the given interval. To do that, use `edb_wait_states_data()`.

2.1.5 Parameters

In addition to the common parameters described previously, each row of the output gives the following:

query

Normalized query text.

2.1.6 Example

The following is a sample output from the `edb_wait_states_queries()` function:

```
edb=# SELECT * FROM edb_wait_states_queries();
__OUTPUT__
- [ RECORD 1 ]+-----
query_id      | 4292540138852956818
query         | SELECT schemaname FROM pg_tables, pg_sleep($1) WHERE schemaname <> $2
ref_start_ts  | 17-AUG-18 11:52:38.698793 -04:00
ref_end_ts    | 18-AUG-18 11:52:38.698793 -04:00
- [ RECORD 2 ]+-----
query_id      | 3754591102365859187
query         | SELECT tablename FROM pg_tables, pg_sleep($1) WHERE schemaname <> $2
ref_start_ts  | 17-AUG-18 11:52:38.698793 -04:00
ref_end_ts    | 18-AUG-18 11:52:38.698793 -04:00
- [ RECORD 3 ]+-----
query_id      | 349089096300352897
query         | SELECT tablename, schemaname FROM pg_tables, pg_sleep($1) WHERE
```

(continues on next page)

(continued from previous page)

```

↪schemaname <> $2
ref_start_ts | 17-AUG-18 11:52:38.698793 -04:00
ref_end_ts   | 18-AUG-18 11:52:38.698793 -04:00

```

2.1.7 edb_wait_states_sessions

This function gives information about the sessions sampled by the BGW:

```

edb_wait_states_sessions(
  IN start_ts timestamptz default -infinity::timestamptz,
  IN end_ts timestamptz default infinity::timestamptz,
  OUT session_id int4,
  OUT <dbname> text,
  OUT <username> text,
  OUT ref_start_ts timestamptz
  OUT ref_end_ts timestamptz
)

```

You can use this function to identify the databases that were connected and the users that started those sessions. For example:

```

SELECT dbname, username, session_id
FROM edb_wait_states_sessions();

```

Similar to `edb_wait_states_queries()`, this function outputs all the sessions logged in session files that contain sessions sampled in the given interval. It doesn't necessarily output only the sessions sampled in the given interval. To identify that, use `edb_wait_states_data()`.

2.1.8 Parameters

In addition to the common parameters described previously, each row of the output gives the following:

dbname

The database to which the session is connected.

username

Login user of the session.

2.1.9 Example

The following is a sample output from the `edb_wait_states_sessions()` function:

```

edb=# SELECT * FROM edb_wait_states_sessions();
__OUTPUT__
- [ RECORD 1 ]+-----
session_id   | 4340
dbname       | edb
username     | enterprisedb
ref_start_ts | 17-AUG-18 11:52:38.698793 -04:00

```

(continues on next page)

(continued from previous page)

```

ref_end_ts | 18-AUG-18 11:52:38.698793 -04:00
- [ RECORD 2 ]+-----
session_id | 4398
dbname     | edb
username   | enterprisedb
ref_start_ts | 17-AUG-18 11:52:38.698793 -04:00
ref_end_ts | 18-AUG-18 11:52:38.698793 -04:00
- [ RECORD 3 ]+-----
session_id | 4410
dbname     | db1
username   | user1
ref_start_ts | 17-AUG-18 11:52:38.698793 -04:00
ref_end_ts | 18-AUG-18 11:52:38.698793 -04:00
- [ RECORD 4 ]+-----
session_id | 4422
dbname     | db2
username   | user2
ref_start_ts | 17-AUG-18 11:52:38.698793 -04:00
ref_end_ts | 18-AUG-18 11:52:38.698793 -04:00

```

2.1.10 edb_wait_states_samples

This function gives information about wait events sampled by the BGW:

```

edb_wait_states_samples(
  IN start_ts timestamptz default -infinity::timestamptz,
  IN end_ts timestamptz default infinity::timestamptz,
  OUT query_id int8,
  OUT session_id int4,
  OUT <query_start_time> timestamptz,
  OUT <sample_time> timestamptz,
  OUT <wait_event_type> text,
  OUT <wait_event> text
)

```

Usually, you don't need to call this function directly.

2.1.11 Parameters

In addition to the common parameters described previously, each row of the output gives the following:

`query_start_time`

The time when the query started in this session.

`sample_time`

The time when wait event data was collected.

`wait_event_type`

The type of wait event on which the session is waiting.

`wait_event`

The wait event on which the session (backend) is waiting.

2.1.12 Example

The following is a sample output from the `edb_wait_states_samples()` function:

```
edb=# SELECT * FROM edb_wait_states_samples();
__OUTPUT__
- [ RECORD 1 ]-----+-----
query_id      | 4292540138852956818
session_id    | 4340
query_start_time | 17-AUG-18 11:56:00.39421 -04:00
sample_time    | 17-AUG-18 11:56:00.699934 -04:00
wait_event_type | Timeout
wait_event     | PgSleep
- [ RECORD 2 ]-----+-----
query_id      | 4292540138852956818
session_id    | 4340
query_start_time | 17-AUG-18 11:56:00.39421 -04:00
sample_time    | 17-AUG-18 11:56:01.699903 -04:00
wait_event_type | Timeout
wait_event     | PgSleep
- [ RECORD 3 ]-----+-----
query_id      | 4292540138852956818
session_id    | 4340
query_start_time | 17-AUG-18 11:56:00.39421 -04:00
sample_time    | 17-AUG-18 11:56:02.700001 -04:00
wait_event_type | Timeout
wait_event     | PgSleep
- [ RECORD 4 ]-----+-----
query_id      | 4292540138852956818
session_id    | 4340
query_start_time | 17-AUG-18 11:56:00.39421 -04:00
sample_time    | 17-AUG-18 11:56:03.700081 -04:00
wait_event_type | Timeout
wait_event     | PgSleep
.
.
.
```

2.1.13 edb_wait_states_purge

The function deletes all the sampled data files (queries, sessions, and wait-event samples) that were created after `start_ts` and aged (rotated) before `end_ts` :

```
edb_wait_states_purge(
  IN start_ts timestamptz default -infinity::timestamptz,
  IN end_ts timestamptz default infinity::timestamptz
)
```

Usually you don't need to run this function. The backend usually purges those according to the retention age. However, if that doesn't happen for some reason, you can use this function.

To find out how long the samples were retained, use `edb_wait_states_data()` .

2.1.14 Example

This code shows the `$PGDATA/edb_wait_states` directory before running `edb_wait_states_purge()` :

```
[root@localhost data]# pwd
/var/lib/edb/as14/data
[root@localhost data]# ls -l edb_wait_states
total 12
-rw----- 1 enterprisedb ... 253 Aug 17 11:56 edb_ws_queries_587836358698793_
↪587922758698793
-rw----- 1 enterprisedb ... 1600 Aug 17 11:56 edb_ws_samples_587836358698793_
↪587839958698793
-rw----- 1 enterprisedb ... 94 Aug 17 11:56 edb_ws_sessions_587836358698793_
↪587922758698793
```

This code shows the `$PGDATA/edb_wait_states` directory after running `edb_wait_states_purge()` :

```
edb=# SELECT * FROM edb_wait_states_purge();
__OUTPUT__
edb_wait_states_purge
-----
(1 row)
```

```
[root@localhost data]# pwd
/var/lib/edb/as14/data
[root@localhost data]# ls -l edb_wait_states
total 0
```

2.2 Deactivating

To disable EDB Wait States from collecting more data, enter:

```
ALTER SYSTEM SET edb_wait_states.enable_collection TO OFF;
SELECT pg_reload_conf();
```

To reenble EDB Wait States, set `edb_wait_states.enable_collection` to ON .

You can check whether the EDB Wait States is collecting data using a `SHOW` command. In the following example, data collection is enabled in EDB Wait States:

```
edb=#SHOW edb_wait_states.enable_collection;
__OUTPUT__
edb_wait_states.enable_collection
-----
on
(1 row)
```