



EDB

EDB Wait States

リリース 1.0

EnterpriseDB Corporation

2023 年 07 月 20 日

目次

第 1 章	Installing EDB Wait States	3
1.1	Linux の場合	3
1.2	Windows	3
1.3	アンインストール	4
第 2 章	Using EDB Wait States	5
2.1	インターフェイスを使用したデータの探索	5
2.1.1	edb_wait_states_data	6
2.1.2	パラメーター	7
2.1.3	例	7
2.1.4	edb_wait_states_queries	8
2.1.5	パラメーター	9
2.1.6	例	9
2.1.7	edb_wait_states_sessions	10
2.1.8	パラメーター	10
2.1.9	例	10
2.1.10	edb_wait_states_samples	11
2.1.11	パラメーター	12
2.1.12	例	12
2.1.13	edb_wait_states_purge	13
2.1.14	例	13
2.2	非アクティブ化	14

EDB Wait States は、待機イベントデータの収集と照会を許可することにより、パフォーマンスとチューニングを分析するためのツールです。PostgreSQL 9.6 で導入された待機イベントは、他のセッションアクティビティと一緒に記録され、セッションが I/O、CPU、IPC、ロック、またはタイムアウトを待機しているかどうかのスナップショットを提供します。この情報のスナップショットは、EDB 待機状態バックグラウンドワーカー（BGW）によって定期的に収集されます。

EDB Wait States インターフェイスを使用すると、待機イベントをサンプリングする時期と期間を制御し、収集したサンプルを `edb_wait_states_data` で抽出してさらに分析できます。このデータを長期にわたって収集することにより、最適化の機会を発見し、パフォーマンスが予想よりも低いときにセッションが待機しているリソースに関する洞察を得ることができます。

第1章 Installing EDB Wait States

1.1 Linux の場合

Linux システムに EDB Wait States をインストールするには：

1. `edb-asxx-server-edb_wait_states` RPM パッケージで EDB Wait States をインストールします。ここで、`xx` は EDB Postgres Advanced Server のバージョン番号です。
1. ワーカーを起動するには、`shared_preload_libraries` パラメーターを使用して `postgresql.conf` ファイルに登録します。例：

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

1. データベースサーバを再起動します。再起動が成功すると、バックグラウンドワーカーはデータの収集を開始します。
1. データを確認するには、次の拡張機能を作成します。

```
CREATE EXTENSION edb_wait_states;
```

1. EDB 待機状態ワーカーを終了するには、`shared_preload_libraries` パラメーターから `$libdir/edb_wait_states` を削除し、データベースサーバを再起動します。

1.2 Windows

Windows システムに EDB Wait States をインストールするには：

1. EDB Modules インストーラーで EDB Wait States モジュールをインストールするには、StackBuilder Plus ユーティリティを呼び出します。画面の指示に従って、EDB Modules のインストールを完了します。
1. ワーカーに登録するには、`postgresql.conf` ファイルを変更して、EDB Wait States ライブラリを `shared_preload_libraries` 構成パラメーターに含めます。パラメーター値には以下を含める必要があります。

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

EDB Wait States のインストールは、`edb_wait_states.dll` ライブラリファイルを次のパスに配置します。

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

1. データベースサーバを再起動して、変更を有効にします。再起動が成功すると、バックグラウンドワーカーが起動し、データの収集を開始します。
1. データを表示するには、次の拡張機能を作成します。

```
CREATE EXTENSION edb_wait_states;
```

インストーラーは、`edb_wait_states.control` ファイルを次のパスに配置します。

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

1.3 アンインストール

EDB Wait States をアンインストールするには：

1. `DROP EXTENSION` コマンドを使用して、`edb_wait_states` 拡張機能を削除します。
2. `postgresql.conf` ファイルを変更し、`shared_preload_libraries` パラメーターから`$libdir/edb_wait_states.dll` を削除します。
1. 変更を適用するには、データベースサーバを再起動します。

第2章 Using EDB Wait States

EDB Wait States が共有プリロードライブラリの 1 つとして登録されている場合、実行中の各セッションを定期的にプローブします。すべてのセッションについて、次のような情報を収集します。

- 接続先のデータベース。
- セッションのログインユーザー。
- そのセッションで実行されているクエリ。
- 待機している待機イベント。

この情報は、`edb_wait_states.directory` パラメーターで指定されたユーザー構成可能なパスとディレクトリフォルダーにある一連のファイルに保存され、`postgresql.conf` ファイルに追加されます。パスは、相対パスではなく、完全な絶対パスである必要があります。

2.1 インターフェイスを使用したデータの探索

EDB Wait States インターフェイスの各機能には、共通の入力パラメーターと出力パラメーターがあります。それらのパラメーターは次のとおりです。

Parameter(s)	Input or output	Description
<i>start_ts</i> and <i>end_ts</i>	Input	これらは一緒に、時間間隔と読み取るデータを指定します。`start_ts`のみを指定した場合、`start_ts`以降のデータが出力されます。`end_ts`のみを指定した場合、`end_ts`までのデータが出力されます。どちらも指定しない場合は、すべてのデータが出力されます。
<i>query_id</i>	Output	正規化されたクエリを識別します。これは、クエリから計算された内部ハッシュコードです。
<i>session_id</i>	Output	セッションを識別します。
<i>ref_start_ts</i> and <i>ref_end_ts</i>	Output	特定のデータポイントを含むファイルのタイムスタンプを提供します。データポイントは、待機イベントのサンプルレコード、クエリレコード、またはセッションレコードの場合があります。

次の例では、異なるユーザーを使用して異なるデータベースに接続された 4 つの異なるセッションで 3 つのクエリが同時に実行されるシナリオを使用します。これら 3 つのクエリは次のとおりです。

```

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

```

2.1.1 edb_wait_states_data

この機能を使用して、BGW によって収集されたデータを読み取ります。

```

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
)

```

この機能を使用して、次のことを確認できます。

- すべてのセッションで指定された期間（start_ts および end_ts で定義）実行されているクエリ、および待機していた待機イベント（ある場合）。例：

```

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

```

- 指定された期間内のセッションの進行状況、つまり、セッション（session_id = 1000000）で実行されたクエリと、クエリが待機した待機イベント。例：

```

SELECT query, wait_event_type, wait_event
FROM edb_wait_states_data(start_ts, end_ts)
WHERE session_id = 1000000;

```

- サンプルが利用可能な期間。例：

```

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

```

2.1.2 パラメーター

前述の一般的なパラメーターに加えて、出力の各行は次のことを示します。

dbname

セッションのデータベース。

username

セッションのログインユーザー。

query

セッションで実行されているクエリ。

query_start_time

クエリが開始された時刻。

sample_time

待ちイベントデータを収集した時刻。

wait_event_type

セッション（バックエンド）が待機している待機イベントのタイプ。

wait_event

セッション（バックエンド）が待機している待機イベント。

2.1.3 例

以下は、`edb_wait_states_data()` 関数の出力例です。

```
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
```

(次のページに続く)

(前のページからの続き)

```

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

この関数は、BGW によってサンプリングされたクエリに関する情報を提供します。例：

```

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
)

```

新しいクエリファイルは定期的に作成されます。特定の間隔に対応する複数のクエリファイルを生成できます。

この関数は、指定された時間間隔と重複するクエリファイル内のすべてのクエリを返します。クエリは、start_ts と end_ts の間でサンプリングされたクエリを含むクエリファイル内のすべてのクエリを提供します。

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

言い換えれば、この関数は、指定された間隔で実行されなかったクエリを出力できます。これを行うには、`edb_wait_states_data()` を使用します。

2.1.5 パラメーター

前述の一般的なパラメーターに加えて、出力の各行は次のことを示します。

`query`

正規化されたクエリテキスト。

2.1.6 例

以下は、`edb_wait_states_queries()` 関数の出力例です。

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

2.1.7 edb_wait_states_sessions

この機能は、BGW によってサンプリングされたセッションに関する情報を提供します。

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

この機能を使用して、接続されたデータベースと、それらのセッションを開始したユーザーを特定できます。例：

```
SELECT dbname, username, session_id
FROM edb_wait_states_sessions();
```

edb_wait_states_queries() と同様に、この関数は、指定された間隔でサンプリングされたセッションを含むセッションファイルに記録されたすべてのセッションを出力します。指定された間隔でサンプリングされたセッションのみを出力するとは限りません。それを識別するには、edb_wait_states_data() を使用します。

2.1.8 パラメーター

前述の一般的なパラメーターに加えて、出力の各行は次のことを示します。

dbname

セッションが接続されているデータベース。

username

セッションのログインユーザー。

2.1.9 例

以下は、edb_wait_states_sessions() 関数の出力例です。

```
edb=# SELECT * FROM edb_wait_states_sessions();
__OUTPUT__
- [ RECORD 1 ]+-----
session_id   | 4340
dbname       | edb
```

(次のページに続く)

(前のページからの続き)

```

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

```

2.1.10 edb_wait_states_samples

この関数は、BGW によってサンプリングされた待機イベントに関する情報を提供します。

```

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
)

```

通常、この関数を直接呼び出す必要はありません。

2.1.11 パラメーター

前述の一般的なパラメーターに加えて、出力の各行は次のことを示します。

`query_start_time`

このセッションでクエリが開始された時刻。

`sample_time`

待ちイベントデータを収集した時刻。

`wait_event_type`

セッションが待機している待機イベントのタイプ。

`wait_event`

セッション（バックエンド）が待機している待機イベント。

2.1.12 例

以下は、`edb_wait_states_samples()` 関数の出力例です。

```
edb=# SELECT * FROM edb_wait_states_samples();
__OUTPUT__
- [ RECORD 1 ]-----+-----
query_id      | 4292540138852956818
session_id    | 4340
query_start_time | 17-AUG-18 11:56:00.39421 -04:00
sample_time    | 17-AUG-18 11:56:00.699934 -04:00
wait_event_type | Timeout
wait_event     | PgSleep
- [ RECORD 2 ]-----+-----
query_id      | 4292540138852956818
session_id    | 4340
query_start_time | 17-AUG-18 11:56:00.39421 -04:00
sample_time    | 17-AUG-18 11:56:01.699003 -04:00
wait_event_type | Timeout
wait_event     | PgSleep
- [ RECORD 3 ]-----+-----
query_id      | 4292540138852956818
session_id    | 4340
query_start_time | 17-AUG-18 11:56:00.39421 -04:00
sample_time    | 17-AUG-18 11:56:02.700001 -04:00
wait_event_type | Timeout
wait_event     | PgSleep
- [ RECORD 4 ]-----+-----
query_id      | 4292540138852956818
```

(次のページに続く)

(前のページからの続き)

```

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

この関数は、start_ts の後に作成され、end_ts の前にエージング（ローテーション）されたすべてのサンプリングデータファイル（クエリ、セッション、および待機イベントサンプル）を削除します。

```

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

```

通常、この関数を実行する必要はありません。バックエンドは通常、保持期間に従ってそれらをパージします。ただし、何らかの理由でそれが起こらない場合、この機能を使用できます。

サンプルが保持された期間を調べるには、edb_wait_states_data() を使用します。

2.1.14 例

このコードは、edb_wait_states_purge() を実行する前に\$PGDATA/edb_wait_states ディレクトリを示しています。

```

[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

```

このコードは、edb_wait_states_purge() を実行した後の\$PGDATA/edb_wait_states ディレクトリを示しています。

```
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 非アクティブ化

EDB 待機状態がより多くのデータを収集しないようにするには、次のように入力します。

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

EDB 待機状態を再度有効にするには、`edb_wait_states.enable_collection` を ON に設定します。

SHOW コマンドを使用して、EDB 待機状態がデータを収集しているかどうかを確認できます。次の例では、EDB 待機状態でデータ収集が有効になっています。

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