



EDB

LiveCompare

リリース 2.4.0

EnterpriseDB Corporation

2022 年 12 月 01 日

目次

第 1 章	パフォーマンス比較	3
第 2 章	ユーザーのセキュリティに関する考慮事項	5
2.1	Release notes	5
2.1.1	Version 2.4	6
2.1.2	Version 2.3	6
2.1.3	Version 2.2	6
2.1.4	Version 2.1	8
2.1.5	Version 2.0	9
2.2	Requirements	11
2.2.1	TPAexec でライブ比較	12
2.3	Supported Technologies	13
2.3.1	PgBouncer のサポート	14
2.4	Command-line Usage	14
2.4.1	コンペアモード	14
2.4.2	再チェックモード	16
2.4.3	コンフリクトモード	17
2.4.4	ドライランモード	17
2.5	Advanced Usage	19
2.5.1	比較中絶	21
2.5.2	比較キー	22
2.5.3	修正する違い	22
2.5.4	差分ログ	23
2.6	BDR Support	23
2.6.1	BDR Witness ノード	26
2.6.2	BDR クラスターの違い	26
2.6.3	BDR の競合	28
2.6.4	混合技術	32
2.7	Oracle Support	32
2.7.1	要件	36
2.7.2	違い	36
2.7.3	BLOB および CLOB データ型	37
2.7.4	互換性のない照合	37
2.7.5	共通ハッシュ	38
2.8	Settings	38
2.8.1	一般設定	38
2.8.2	初期接続	45
2.8.3	出力接続	46
2.8.4	データ接続	46
2.8.5	テーブルフィルター	47

2.8.6	行フィルター	49
2.8.7	カラムフィルター	50
2.8.8	比較キー	51
2.8.9	競合フィルター	52
2.9	Licenses	52
2.9.1	TQDM	52
2.9.2	cx_Oracle	53
2.9.3	Psycpg2	56
2.9.4	表にする	57
2.9.5	OmniDB	57

LiveCompare は、任意の数のデータベースを比較して、それらが同一であることを確認するように設計されています。このツールは任意の数のデータベースを比較し、比較レポート、違いのリスト、および便利な DML スクリプトを生成するため、ユーザーはオプションで DML を適用して任意のデータベースの不整合を修正できます。

デフォルトでは、比較セットにはデータベース内のすべてのテーブルが含まれます。LiveCompare では、複数のテーブル（複数のワーカープロセス）を同時にチェックでき、いくつかのテーブルまたはテーブル内の行のセクションのみをチェックできるように高度に構成できます。

各データベース比較は「比較セッション」と呼ばれます。プログラムが初めて起動すると、新しいセッションが開始され、テーブルごとの比較が開始されます。スタンドアロンモードでは、すべてのテーブルが比較されると、プログラムが停止してすべてのレポートを生成します。LiveCompare は、コンテキスト情報を失うことなく停止および開始できるため、都合の良いときに実行できます。

各テーブル比較操作は「比較ラウンド」と呼ばれます。テーブルが大きすぎる場合、LiveCompare はテーブルを多重比較ラウンドに分割し、同時に他のワーカーが実行している他のテーブルとともに実行します。

スタンドアロンモードでは、テーブルの最初の比較ラウンドはテーブルの先頭（既存の最も古い PK）からテーブルの最後（既存の PK）まで開始されます。ラウンドが開始された後に挿入された新しい行は無視されます。LiveCompare は、各テーブルから最小および最大 PK を取得するために PK 列を並べ替えます。ソートできない PK 列ごとに、LiveCompare はそのコンテンツを `string` にキャストします。PostgreSQL では `::text` を使用して、Oracle では `to_char` を使用して実現します。

比較アルゴリズムを実行する場合、各ワーカーには $N+1$ 個のデータベース接続が必要です。N は比較されるデータベースの数です。追加に必要な接続は、プログラムキャッシュも保持される出力/レポートデータベースへの接続であるため、ユーザーは比較セッションを停止/再開できます。

比較アルゴリズムによって検出された違いは、ユーザーが後で都合の良いときに手動で再確認できます。これは、レプリケーションの一貫性チェックを許可するために実行することをお勧めします。違いを再チェックすると、レプリケーションがその特定の行に追いつき、違いが存在しない可能性があるため、違いは削除されるか、永続的としてマークされます。

実行の最後に、プログラムは DML スクリプトを生成して、ユーザーがそれを確認し、違いを 1 つずつ修正するか、単に DML スクリプト全体を適用してすべての永続的な違いを修正できます。

LiveCompare を使用して、行レベルで論理データの整合性を保証できます。たとえば、次のシナリオの場合。

- データベース技術の移行（Oracle x Postgres）；
- サーバーの移行またはアップグレード（古いサーバー×新しいサーバー）；
- 物理レプリケーション（プライマリ×スタンバイ）；
- フェイルオーバーインシデントの後、たとえば、新しいプライマリデータを古い分離されたプライマリデータと比較するため。
- フェイルオーバー後に予期しないスプリットブレイン状況が発生した場合。古いプライマリが適切にフェンスされておらず、アプリケーションがデータを書き込んだ場合、LiveCompare を使用して、古いプライマリに存在し、新しいプライマリに存在しないデータを正確に知ることができます。必要に応じて、DBA は LiveCompare が生成する DML スクリプトを使用して、これらのデータを新しいプライマリに適用できます。
- 論理複製。Postgres ネイティブ論理レプリケーション、pglogical、BDR の 3 種類の論理レプリケーションテクノロジーがサポートされています。

第1章 パフォーマンス比較

LiveCompare は実稼働システムでの使用に最適化されており、後述するチューニング用のさまざまなパラメーターがあります。比較ラウンドは読み取り専用のワークロードです。ユースケースの例では、4つの接続と4つのワーカーを使用して9分17秒で6つのテーブルの43,109,165行を比較し、毎秒約77,000行、または4時間未満で10億行のパフォーマンス比較を得ました。

上記のユースケースは、一般的なユースケースと見なすことができます。低負荷、テスト、移行、およびその他の特定のシナリオでは、サーバー側のカーソルを使用するように `data_fetch_mode` 設定を変更することで速度を向上できる場合があります。私たちの実験では、さまざまな種類のサーバーサイドカーソルを使用すると、小規模または大規模なテーブルを含むユースケースでパフォーマンスが向上します。

第2章 ユーザーのセキュリティに関する考慮事項

PostgreSQL 13 以前の場合、LiveCompare には比較されるすべてのデータを読み取ることができるユーザーが必要です。PostgreSQL 14 では、LiveCompare に使用できる新しいロール `pg_read_all_data` が導入されました。

`logical_replication_mode = bdr` の場合、LiveCompare には `bdr_superuser` ロールが付与されているユーザーが必要です。`logical_replication_mode = pglogical` の場合、LiveCompare には `pglogical_superuser` ロールが付与されているユーザーが必要です。

BDR で DML スクリプトを適用するには、すべての発散接続（潜在的にすべてのデータ接続）で、`bdr.xact_replication` を無効にするために `bdr_superuser` が付与されている必要があります。

BDR が使用されている場合、LiveCompare はすべての固定行を `bdr_local_only_origin` と呼ばれる複製起点に関連付けます。LiveCompare は、はるかに過去のトランザクション日時で DML を適用するため、データベースで実行されている実際の DML との BDR 競合がある場合、LiveCompare DML は常に競合を失います。

`difference_fix_start_query` のデフォルト設定では、適用スクリプトのトランザクションはロールを表の所有者に変更し、データベースユーザーが悪意のあるトリガーを作成して修正を適用するロールにアクセスするのを防ぎます。その結果、ダイバージェント接続のユーザーには、ロールをテーブル所有者に切り替える機能が必要です。

2.1 Release notes

LiveCompare のドキュメントには、マイナーリリースとパッチを含む LiveCompare 2 の最新バージョンが記載されています。このセクションのリリースノートには、各リリースの新機能に関する情報が記載されています。

Version	Release Date
<i>Version 2.4</i>	Nov 29 2022
<i>Version 2.3</i>	Aug 15 2022
<i>Version 2.2</i>	Jun 14 2022
<i>Version 2.1</i>	Mar 31 2022
<i>Version 2.0</i>	Feb 15 2022

2.1.1 Version 2.4

LiveCompare 2.4 には、次の新機能、機能強化、バグ修正、およびその他の変更が含まれています。

Type	Description
Bug fix	複合主キーを持つテーブルで QueryTablePKColumns の実行が遅くなる問題を修正しました。
Bug fix	LiveCompare が時間列を正しく処理していない問題を修正しました。
Bug fix	一重引用符またはアポストロフィを含む接続文字列が正しく処理されない問題を修正しました。
Bug fix	`--recheck` オプションが一部の状況で未処理の例外をスローする問題を修正しました。

2.1.2 Version 2.3

LiveCompare 2.3 には、次の新機能、機能強化、バグ修正、およびその他の変更が含まれています。

Type	Description	ID
Enhancement	IBM Power (ppc64le) での RedHat Enterprise Linux (RHEL) 8 のサポート。	
Enhancement	IBM Power (ppc64le) での SLES 12 SP5 のサポート。	
Enhancement	IBM Power (ppc64le) での SLES 15 SP3 のサポート。	
Enhancement	x86_64 での Ubuntu 22.04 (Jammy) のサポート。	
Enhancement	x86_64 での Debian 11 (ブルズアイ) のサポート。	

2.1.3 Version 2.2

LiveCompare 2.2 には、次の新機能、機能強化、およびバグ修正が含まれています。

Type	Description	ID
New feature	<code>--dry-run</code> 実行モードが追加されました。これにより、ユーザーは比較セッションを実行せずに <code>.ini</code> ファイルを検証し、いくつかの有用な比較情報を表示できます。詳細については、 <code>:ref:</code> ドライランモード<ドライランモード>を参照してください。	LIV-142, RT78462
Feature	比較セッション中に受信した中断メッセージは、時系列で出力されます。	
Feature	テクノロジー、バージョン、接続がタイブレーカーか信頼できるソースか、到達可能かどうかを含む接続のリストを表示します。	
Feature	<code>`Table Filter`</code> が構成されている場合は表示します。	
Feature	比較に含まれるテーブルをリストします。これは、 <code>Table Filter`</code> を適用した後のすべての接続に存在する共通テーブルのリストです。各テーブルについて、該当する場合、 <code>`Column Filter`</code> 、 <code>`Row Filter`</code> および <code>`Comparison Key`</code> を示します。	
Enhancement	x86_64 での SLES 12 のサポート。	LIV-112
Enhancement	x86_64 での SLES 15 のサポート。	LIV-113
Enhancement	Postgres <code>`kwlist`</code> ソースコードに従って予約語のリストを更新しました。	LIV-141, RT80745
Enhancement	メインの進行状況バーのパーセンテージは、整数ではなく浮動小数点数を使用しています。以前は、数値は切り上げられました。	LIV-138, RT80745
Enhancement	EPAS で <code>`edb_redwood_date`</code> が有効になっている場合、 <code>`date`</code> 列は <code>`timestamp`</code> 列として作成されます。この型の不一致は、以前に共通ハッシュの不一致として検出され、行全体の比較がトリガーされました。これらの場合に <code>`edb_redwood_date`</code> 設定を確認することにより、共通ハッシュ（完全な行よりも高速）でこの不一致を処理することにより、パフォーマンスが向上します。	LIV-153
Enhancement	ハッシュ不一致ログメッセージを WARNING から DEBUG に降格。	LIV-145
Enhancement	ログには、メッセージに <code>application_name</code> が含まれます。	LIV-141
Enhancement	<code>`data_fetch_mode = prepared_statements`</code> （デフォルト）の場合、Postgres または EPAS で <code>`Row Filter`</code> の現在のタイムスタンプを使用する動作を文書化しました。この場合、 <code>`data_fetch_mode = server_side_cursors_with_hold`</code> または <code>`server_side_cursors_without_hold`</code> も設定する必要があります。	LIV-155

次のページに続く

表 4 – 前のページからの続き

Type	Description	ID
Enhancement	検出された相違の数が増加した場合に適用 DML スクリプトを生成する際のパフォーマンスが向上しました。	LIV-159
Bug fix	配列列が比較キーと見なされると、テーブルでの比較が例外により中止される問題を修正しました。	LIV-38, LIV-154, RT81758
Bug fix	ハートビートによってのみ、比較ラウンド後に発散および処理された行の数が適切に更新されない問題は解決されました。この場合、行は古く、最後のハートビート以降の位置のみを示しています。比較にかかったテーブルが `min_time_between_heart_beats` よりも短い場合、常にゼロを示します。	LIV-149, RT75805
Bug fix	未処理の例外により比較ワーカーがハングする可能性があるコーナーケースを修正しました。	LIV-140, RT80745
Bug fix	Oracle と Postgres を比較するときに誤検出を回避するための 10 進値の正規化。	LIV-153
Bug fix	`comparison_algorithm = block_hash` と `buffer_size = 1` で、相違が見つかった場合、カーソル前進アルゴリズムの問題が原因で比較が進められないコーナーケースを修正しました。	LIV-150
Bug fix	複数のフィルターセクションで同じテーブルを構成できるようにします。	LIV-156

2.1.4 Version 2.1

LiveCompare 2.1 には、次の新機能、バグ修正、およびその他の変更が含まれています。

Type	Description	ID
Feature	EDB Postgres Distributed 4 のサポート。	LIV-131
Feature	ハートビートごとに比較の進行状況をログに記録するように LiveCompare に指示する新しい設定 `min_time_between_heart_beats` は、`INFO` ログレベルを使用してデフォルトで 30 秒に設定されます。	LIV-128
Feature	新しい設定 <i>comparison_cost_limit</i> および <i>comparison_cost_delay</i> 。0 より大きい場合、各ワーカーに `comparison_cost_limit` 数の行を処理した後に `comparison_cost_delay` 秒 (たとえば、0.5) 昼寝をするように指示します。	LIV-16

次のページに続く

表 5 – 前のページからの続き

Type	Description	ID
Change	`parallel_chunk_rows`のデフォルト値は`0`に設定されています。最近の調査により、一般的なユースケースのパフォーマンスが低下することが判明したため、デフォルトでテーブル分割を無効にします。詳細については、:ref:`コンペアモード<コンペアモード>`を参照してください。	LIV-130
Change	`CanAdvanceCursors`メソッドからの処理された行数に関するログメッセージを`DEBUG`に降格しました。	LIV-129
Bug fix	`ORA-01830`で失敗する`timestamp(6)`データ型の Oracle と Postgres の比較の問題を修正しました。	LIV-127
	該当なし	

2.1.5 Version 2.0

LiveCompare 2.0 には、次の新機能、機能強化、バグ修正、およびその他の変更が含まれています。

Type	Description	ID
New feature	`Comparison Key`と呼ばれる設定の新しいセクションにより、ユーザーはテーブルごとにカスタム比較キー（列のリスト）を定義できます。これは、主キーまたは一意のインデックスのないテーブルに役立ちます。詳細については、:ref:`比較キー<比較キー>`を参照してください。	LIV-5
Feature	テーブルにユーザー定義の比較キーまたは主キーがない場合、LiveCompare はテーブルの一意のインデックスを使用しようとします。テーブルは複数の一意のインデックスを持つことができるため、LiveCompare は列が null 許容でないインデックスを使用することを好みます。不可能な場合、LiveCompare は列の少ない最初の一意のインデックスを使用しようとします。一意のインデックスがない場合、LiveCompare はテーブルのすべての列を比較キーとして使用しようとします。LiveCompare は一意のインデックスからの null 許容列を無視しないことに注意してください。	LIV-39

次のページに続く

表 6 – 前のページからの続き

Type	Description	ID
Feature	テーブルのすべての列を比較キーとして使用する 場合、LiveCompare は null 許容列を無視して、 すべての列を検討する際の誤検出を回避するよ うになりました。この動作は、 <i>ignore_nullable = false</i> を設定することで無効にできます。	LIV-96
Feature	ユーザー定義の比較キーまたはテーブルのすべ ての列を比較キーとして使用する場合、Live- Compare は一意性を強制するかどうかを確認す るようになりました。列セットを使用してい る時に値が重複している場合、LiveCompare は テーブルでの比較を中止します。この動作は、 <i>check_uniqueness_enforcement = false</i> を設定する ことで無効にできます。	LIV-119
Enhancement	Oracle 21c のサポートが追加されました。	LIV-47
Enhancement	Oracle 12c 以降では、LiveCompare は既に共通 のハッシュを使用して <code>`comparison_algorithm = block_hash`</code> とワーカーを使用できます。これは、 Oracle 12c で導入された <i>standard_hash()</i> 関数 を使用して Oracle 側で実行されます。現在、 LiveCompare は、ユーザーが <code>`sys.dbms_crypto`</code> Oracle システムパッケージに対する <code>`EXE- CUTE`</code> 特権を持っている場合、同等の機能 <code>`sys.dbms_crypto.hash()`</code> を使用して、Oracle 11g で <code>`comparison_algorithm = block_hash`</code> および <code>`comparison_algorithm = row_hash`</code> も許可します。	LIV-79
Enhancement	LiveCompare スキーマをレプリケーション対応 (EDB Postgres 分散、pglogical またはネイティブ 論理レプリケーション) に追加できるようにな りました。	LIV-42
Enhancement	LiveCompare は、PostgreSQL 14 の新しい <code>`pg_read_all_data`</code> ロールを使用できます。	LIV-73
Enhancement	データベースバージョンがサポートされてい ない場合は、適切なメッセージで中止します。	
Change	パッケージは <code>`2ndq-livecompare`</code> から <code>`edb- livecompare`</code> に名前が変更されました。	
Change	実行可能ファイルは、 <code>`2ndq-livecompare`</code> から <code>`livecompare`</code> に名前が変更されました。	
Bug fix	DML 生成時のエラーを回避するには、ユニコー ドシーケンス <i>u0000</i> を適切に引用します。	LIV-98
Bug fix	テーブルが複数のラウンド部分に分割された ときに、合計行数が誤って表示される問題を修正 しました。	LIV-14

次のページに続く

表 6 – 前のページからの続き

Type	Description	ID
Bug fix	Postgres の空の `bytea` と比較したときに、Oracle の空の `BLOB` が誤検知を生成していた問題を修正しました。	LIV-103
Bug fix	接続の問題が原因で比較セッション全体が中止される問題を修正しました。これで、LiveCompare は再接続して、可能な場合は比較を続行できます。	LIV-84
Bug fix	相違がある場合の再チェックモードでの未処理の例外を修正しました。	LIV-107
Bug fix	接続にテーブルが存在せず、`logical_replication_mode` が無効になっている場合、テーブルの比較が中止されない問題を修正しました。	LIV-108
Bug fix	`timestamp` データ型のフィールドが Oracle と Postgres の間で常に一致しないハッシュを生成していた問題を修正しました。	
Bug fix	無視された列が共通のハッシュでまだ考慮されている問題を修正しました。	

2.2 Requirements

LiveCompare には以下が必要です。

- Python 3.6 または 3.7
- PostgreSQL / EDB Postgres Extended 9.5+ / EPAS 11+（出力接続）
- PostgreSQL / EDB Postgres Extended 9.4+ / EPAS 11+ または Oracle 11g+（比較されるデータ接続）

LiveCompare には、Debian 10 以降、Ubuntu 16.04 以降、SLES 12 SP5 および 15 SP3、または CentOS/RHEL/RockyLinux/AlmaLinux 7 以降が必要です。

LiveCompare は EnterpriseDB products/livecompare リポジトリからインストールできます。詳細については、次をご覧ください。

https://techsupport.enterprisedb.com/customer_portal/sw/livecompare/

LiveCompare は次の上にインストールします。

- `python3` パッケージによって提供される、Ubuntu、Debian、および CentOS / RHEL 8 用の最新の Python バージョン。または
- `python-36` パッケージによって提供される CentOS / RHEL 7 用の Python 3.6。

CentOS / RHEL ディストリビューションでは、LiveCompare に EPEL リポジトリも必要です。詳細については、次をご覧ください。

<https://fedoraproject.org/wiki/EPEL>

特に CentOS / RHEL バージョン 7 では、Python コンポーネント `tqdm` が古すぎます (<4.16.0)。LiveCompare を実行しているユーザーの `pip` または `pip3` を使用して、最新の `tqdm` をインストールできます。

```
pip install --user tqdm --upgrade
```

Oracle データベースに対して LiveCompare を実行する場合、Oracle Instant Client をインストールする必要があります。追加情報については、

Oracle Support Requirements を参照してください。

2.2.1 TPAexec でライブ比較

次の TPAexec のサンプル構成を使用して、LiveCompare および PostgreSQL 11 でサーバーを構築できます。

```
- --
architecture: M1
cluster_name: livecompare_m1
cluster_tags: {}

cluster_vars:
  postgres_coredump_filter: 0xff
  postgres_version: 13
  postgresql_flavour: postgresql
  repmgr_failover: manual
  tpa_2q_repositories:
    - products/livecompare/release
  packages:
    common:
      - edb-livecompare
  use_volatile_subscriptions: true

locations:

- Name: main

instance_defaults:
  image: tpa/rocky
  platform: docker
  vars:
    ansible_user: root

instances:

- Name: livem1node1
  location: main
```

(次のページに続く)

(前のページからの続き)

```

node: 1
role: primary
published_ports:
  - 5401:5432

- Name: livem1node2
  location: main
  node: 2
  role: replica
  upstream: livem1node1
  published_ports:
    - 5402:5432

```

TPAexec の詳細については、次を参照してください。

https://techsupport.enterprisedb.com/customer_portal/sw/tpa/

2.3 Supported Technologies

LiveCompare は、PostgreSQL、BDR、Oracle などのテクノロジーのリストに接続してデータを比較できます。

LiveCompare には 3 種類の接続があります。

- **Initial** (オプション) : pglogical または BDR 接続に関するメタデータを取得するために使用されます。データ接続が pglogical または BDR、および `replication_sets` または `node_name` 設定が使用される場合に必要です。 `logical_replication_mode = pglogical` または `logical_replication_mode = bdr` が必要です。 pglogical または BDR 対応のデータベースである必要があります。
- **データ** : ツールがデータ比較を実行するために接続する実際のデータベース接続。リストの最初の接続は、Table Filter および Row Filter を解決するために使用され、初期接続と組み合わせて BDR ノードに関する情報を収集するためにも使用されます。 `logical_replication_mode = bdr` および `all_bdr_nodes = on` の場合、LiveCompare は Initial Connection と同じ BDR クラスターの一部であるすべての BDR ノードを検討します。この場合、データ接続を個別に定義する必要はありません。比較とコンセンサスの決定は行ごとに機能するため、修正はすべてのデータ接続に適用される可能性があります。
- **出力** (必須) : LiveCompare は `livecompare` と呼ばれるスキーマ、いくつかのテーブルとビューを作成します。これは、比較セッションに関する進行状況とレポートデータを維持するために必要です。PostgreSQL または 2ndQPostgres 接続である必要があります。

以下に、サポートされているテクノロジーのバージョンと詳細、および LiveCompare で使用できるコンテキストを示します。

Technology	Versions	Connections
PostgreSQL	10, 11, 12, 13 and 14	Data and/or Output
EDB PostgreSQL Extended	10, 11, 12, 13 and 14	Data and/or Output
EDB PostgreSQL Advanced (EPAS)	11, 12, 13 and 14	Data and/or Output
pglogical	2 and 3	Initial, Data and/or Output
BDR	1, 2, 3 and 4	Initial, Data and/or Output
Oracle	11g, 12c, 18c, 19c and 21c	A single Data connection

2.3.1 PgBouncer のサポート

LiveCompare は PgBouncer を介してノードに対して使用できますが、LiveCompare は PostgreSQL で準備されたステートメントを使用するため、`pool_mode` が `transaction` または `statement` の場合は使用できません。

2.4 Command-line Usage

2.4.1 コンペアモード

プロジェクトで使用する `/etc/livecompare/template*.ini` をコピーし、必要に応じて調整します（以下の `Settings` セクションを参照）。

```
cp /etc/livecompare/template_basic.ini my_project.ini

livecompare my_project.ini
```

LiveCompare の実行中に、`N+1` 進行状況バーが表示されます。`N` はプロセス数です（設定でプロセス数を指定できます）。最初の進行状況バーは全体的な実行を示し、他の進行状況バーは特定のプロセスによって処理されている現在のテーブルを示します。

各テーブルに表示される情報は、左から右に次のとおりです。

- プロセス数
- テーブル名
- ステータス。比較ラウンドの後に現在のテーブルチャンクが続く場合があります（`p1/1` は、テーブルが分割されなかったことを意味します）。ステータスが `setup` の場合、テーブルが分析されていることを意味します（行数を確認し、必要に応じて分割します）
- 処理された行数
- この比較ラウンドで考慮される合計行数
- 経過時間
- 推定所要時間

- 1 秒あたりのレコードの速度

テーブル分割が有効になっている場合 (`parallel_chunk_rows > 0`)、テーブルに `parallel_chunk_rows` 設定よりも多くの行がある場合、ハッシュ関数を使用して各行を考慮するジョブが決定されます。これにより、個々の比較が遅くなる可能性があります、比較全体では、指定されたテーブルの並列処理が役立つ場合があります。

プログラムの実行中、`Ctrl-c` を押すことでいつでもキャンセルできます。次のようなメッセージが表示されます。

```
Manually stopping session 6... You can resume the session with:
```

```
livecompare my_project.ini 6
```

重要：LiveCompare がバックグラウンドで実行されている場合、または別のシェルで実行されている場合でも、ソフトに停止できます。マスタープロセスの PID を、セッションフォルダー（例では `lc_session_6`）に、`livemaster.pid` という名前のファイルに保持します。次に、`kill -2 <PID>` を呼び出してソフトに停止できます。

次に、以前にキャンセルされたセッションをいつでも再開できます。

```
livecompare my_project.ini 6
```

プログラムが終了して、矛盾が見つからなかった場合、次のような出力が表示されます。

```
Saved file lc_session_5/summary_20190514.out with the complete table summary.
You can also get the table summary by connecting to the output database and executing:
select * from livecompare.vw_table_summary where session_id = 5;

Elapsed time: 0:02:10.970954
Processed 3919015 rows in 6 tables using 3 processes.
Found 0 inconsistent rows in 0 tables.
```

ただし、矛盾が見つかった場合、出力は次のようになります。

```
Comparison finished, waiting for remaining difference checks...
```

```
Outstanding differences:
```

```
+-----+-----+-----+-----+-----+
| session_id | table_name      | elapsed_time    | num_total_rows | num_
| processed_rows | num_differences | max_num_ignored_columns |
+-----+-----+-----+-----+-----+
|          6 | public.categories | 00:00:00.027864 |          18 |
|         18 |                   |                  |           4 |
+-----+-----+-----+-----+-----+
|          6 | public.categories | 00:00:00.027864 |          18 |
|         18 |                   |                  |           4 |
+-----+-----+-----+-----+-----+
```

(次のページに続く)

```
Saved file lc_session_6/summary_20200129.out with the complete table summary.
You can also get the table summary by connecting to the output database and executing:
select * from livecompare.vw_table_summary where session_id = 6;

Elapsed time: 0:00:50.149987
Processed 172718 rows in 8 tables from 3 connections using 2 workers.
Found 4 inconsistent rows in 1 tables.

Saved file lc_session_6/differences_20200129.out with the list of differences per
→table.
You can also get a list of differences per table with:
select * from livecompare.vw_differences where session_id = 6;
Too see more details on how LiveCompare determined the differences:
select * from livecompare.vw_consensus where session_id = 6;

Script lc_session_6/apply_on_the_first_20200129.sql was generated, which can be
→applied to the first connection and make it consistent with the majority of
→connections.
You can also get this script with:
select difference_fix_dml from livecompare.vw_difference_fix where session_id = 6 and
→connection_id = first;
```

2.4.2 再チェックモード

BDR 環境では、最終的な一貫性のためにレプリケーションが追いつくため、BDR が検出した相違が後で存在しない場合があります。いくつかの要因によっては、レプリケーションラグが原因で LiveCompare が誤検出を報告する場合があります。

それを克服するために、レプリケーションの遅延が減少するか、データが既に追いついたときに、ユーザーは以前に見つかった違いのみで再チェックを実行できます。この実行モードは「再チェック」と呼ばれ、次のように実行できます。

```
livecompare my_project.ini 6 --recheck
```

このモードでは、LiveCompare は個別の再チェックログを生成し、lc_session_X ディレクトリに既に存在するすべてのレポートを更新します。

重要： compare を再開するか、recheck で実行する場合、LiveCompare は設定と接続の属性がセッションが作成されたときと同じかどうかを確認します。相違が見つかった場合、適切なメッセージで実行を終了します。

2.4.3 コンフリクトモード

LiveCompare を `conflicts` モードで実行するには、次のように呼び出す必要があります。

```
livecompare my_project.ini --conflicts
```

`conflicts` モードの詳細については、BDR サポートの章を確認してください。

2.4.4 ドライランモード

たとえば、次の INI ファイルがあります。

```
[General Settings]
logical_replication_mode = off
difference_tie_breakers = first

[First Connection]
dsn = dbname=testb

[Second Connection]
dsn = dbname=testdb2

[Third Connection]
dsn = dbname=testdb3

[Output Connection]
dsn = dbname=liveoutput

[Table Filter]
schemas = schema_name = public
```

Output Connection（LiveCompare キャッシュデータベース）の DSN が正しくないため、LiveCompare の実行は最初に次のエラーで失敗します。

```
Output connection is not reachable.
```

これを修正した後、Output Connection に到達できるようになりましたが、データ接続の 1 つだけが正しく設定されていると想像してください。LiveCompare は次で再び失敗します。

```
At least two reachable connections are required.
Following connections are unreachable: first, third.
Following connections are reachable: second.
```

LiveCompare は、使用可能な少なくとも 2 つのデータ接続で比較を開始できます。したがって、先に進んで Third Connection を修正しますが、LiveCompare はまだ次のように失敗します。

A difference_tie_breakers host **is not** a reachable connection: first.

これは、`difference_tie_breakers = first`を設定しており、タイブレーカーまたは信頼できるソースとして設定された接続に到達できる必要があるために発生します。

これらの問題をすべて修正した後、LiveCompare は比較を開始できます。

ただし、最初から比較を設定する場合、LiveCompare が構成エラーで中断するかどうかを事前に確認でき、それ以降の中断チェックはすべて LiveCompare によって実行される順序で表示されます。

これは`--dry-run` モードで可能です。

- 構成の問題が原因で発生するすべての実行中止を出力します。
- 接続のリストを、到達可能かどうかを含む詳細とともに出力します。
- Table Filter を印刷します。
- Table Filter を適用した後、到達可能な接続に共通するテーブルのリストを出力します。

上記の INI ファイルと、到達不能な接続に関するすべての構成エラーを前提とした 1 つの出力例を次に示します。

```
$ livecompare test.ini --dry-run
EnterpriseDB LiveCompare 2.2.0, dry-run mode

Output connection is not reachable.

At least two reachable connections are required.
Following connections are unreachable: first, third.
Following connections are reachable: second.

A difference_tie_breakers host is not a reachable connection: first.

Connections

+-----+-----+-----+-----+-----+-----+
↪+-----+-----+-----+-----+
| ID      | Technology | Version | BDR Version | Pglogical Version | Initial |
↪| Tie Breaker | Source of Truth | Reachable |
|-----+-----+-----+-----+-----+-----+
↪+-----+-----+-----+-----+
| second | postgresql | 110015 | -           | -                 | False |
↪| False      | False      | True   |
| first  | postgresql | -      | -           | -                 | False |
↪| True       | False      | False  |
| third  | postgresql | -      | -           | -                 | False |
↪| False      | False      | False  |
| output | postgresql | -      | -           | -                 | -     |
↪| -          | -          | False  |
```

(次のページに続く)

(前のページからの続き)

Table Filter				
publications =				
replication_sets =				
schemas = schema_name = public				
tables =				
Tables				
Table Name	Row Filter	Column Filter	Custom Comparison Key	
public.categories	-	-	-	
public.cust_hist	-	-	-	
public.customers	-	-	-	
public.departments	-	-	-	
public.dept_emp	-	-	-	
public.dept_manager	-	-	-	
public.employees	-	-	-	
public.inventory	-	-	-	
public.orderlines	-	-	-	
public.orders	-	-	-	
public.products	-	-	-	
public.reorder	-	-	-	
public.salaries	-	-	-	
public.tbl	-	-	-	
public.titles	-	-	-	

2.5 Advanced Usage

LiveCompare の実行が終了すると、作業ディレクトリに `lc_session_<session_id>` というフォルダーが作成されました。このフォルダーには、次のファイルが含まれています。

- `lc_<execution_mode>_<current_date>.log` : セッションのログファイル。
- `summary_<current_date>.out` : 処理されたすべてのテーブルのリストを表示し、各テーブルについて、LiveCompare がテーブルを処理するのにかかった時間、処理された行の合計数と行数、テーブルで見つかった違いの数、およびまた、無視される列の最大数（ある場合）。

完全な概要を取得するには、出力データベースに対して次のクエリを実行することもできます。

```
select *
from <output_schema>.vw_table_summary
where session_id = <session_id>;
```

- `differences_<current_date>.out` : 違いがある場合、このファイルには各違いに関する有用な情報が表示されます。違いがない場合、このファイルは生成されません。

たとえば、差分リストは次のようになります。

```
+-----+-----+-----+-----+
↪+
| table_name      | table_pk_column_names | difference_pk | difference_status |
↪|
|-----+-----+-----+-----+
↪|
| public.categories | category              | (7) | P |
↪|
| public.categories | category              | (10) | P |
↪|
| public.categories | category              | (17) | P |
↪|
| public.categories | category              | (18) | P |
↪|
+-----+-----+-----+-----+
↪+
```

すべての詳細を含む相違点の完全なリストを取得するには、出力データベースに対して次のクエリを実行することもできます。

```
select *
from <output_schema>.vw_differences
where session_id = <session_id>;
```

LiveCompare コンセンサスがどのデータベースが発散しているかを判断する方法を理解するために、ビュー `vw_consensus` はコンセンサスアルゴリズムの詳細を提供できます。

```
select *
from <output_schema>.vw_consensus
where session_id = <session_id>;
```

- `apply_on_the_first_<current_date>.sql` : 違いがある場合、このファイルは **first** データベースに適用されて、他のすべてのデータベースの一貫性を保つための DML コマンドを表示します。たとえば、上記の違いの場合、このスクリプトは次のようになります。

```
BEGIN;

DELETE FROM public.categories WHERE (category) = 7;
```

(次のページに続く)

(前のページからの続き)

```
UPDATE public.categories SET categoryname = $lc1$Games Changed$lc1$ WHERE (category)
↳= 10;
INSERT INTO public.categories (category,categoryname) VALUES (17, $lc1$Test 1$lc1$);
INSERT INTO public.categories (category,categoryname) VALUES (18, $lc1$Test 2$lc1$);

COMMIT;
```

LiveCompare はこのスクリプトを自動的に生成します。 **first** データベースの不整合を修正するには、 **first** データベースでスクリプトを実行するだけです。

LiveCompare は、一貫性のないデータを持つデータベースごとに同様の `apply_on_*.sql` スクリプトを生成します。

2.5.1 比較中絶

比較セッションを開始する前に、LiveCompare はすべての接続を試行します。到達可能な接続数が 2 以上でない場合、LiveCompare は適切なエラーメッセージを表示してセッション全体を中止します。少なくとも 2 つの接続が到達可能な場合、LiveCompare は比較セッションを続行します。すべての接続について、LiveCompare はフラグ `connection_reachable` をキャッシュデータベースの `connections` テーブルに書き込みます。

到達可能なすべての接続について、LiveCompare はデータベーステクノロジーと設定 `logical_replication_mode` についていくつかの健全性チェックを行います。いずれかの健全性チェックが失敗した場合、LiveCompare は適切なエラーメッセージを表示して比較を中止します。

LiveCompare は、到達可能なすべての接続で使用可能なテーブルを考慮し、`Table Filter`. 特定のテーブルが少なくとも 2 つの接続に存在しない場合、その特定のテーブルでの比較は中止されます。

LiveCompare は最初にすべてのテーブルからメタデータを収集します。この手順を「セットアップ」と呼びます。セットアップ中にエラーが発生した場合（たとえば、ユーザーが特定のテーブルにアクセスできない場合）、それは「セットアップエラー」と呼ばれます。`abort_on_setup_error` が有効になっている場合、LiveCompare は比較セッション全体を中止し、プログラムはエラーメッセージで終了します。それ以外の場合、エラーが発生した特定のテーブルのみがテーブル比較を中止し、LiveCompare は次のテーブルに進みます。

各テーブルに対して、LiveCompare はテーブル比較を開始します。最初に、LiveCompare は到達可能なすべての接続でテーブル定義を確認します。テーブルの列と列のデータ型が異なる場合、LiveCompare は `column_intersection` を適用します。比較する列がない場合、LiveCompare はテーブルの比較を中止します。

2.5.2 比較キー

比較されるテーブルごとに、テーブルのメタデータを収集するときに、LiveCompare は次のルールに従って、テーブルの比較で使用される **Comparison Key** をビルドします。

- 1-ユーザーが構成した場合はカスタム **Comparison Key** を使用します。または
- 2-利用可能な場合は PK を使用します。または
- 3-テーブルに **UNIQUE** インデックスがある場合：すべての **NOT NULL** 列を持つ **UNIQUE** インデックスの中で、より少ない列で **UNIQUE** インデックスを使用します。または
- 4-上記のいずれも不可能な場合は、すべての **NOT NULL** 列を比較キーとして使用してみてください (**ignore_nullable = false** の場合は **NULL** 列も検討できます)。

上記の戦略 1 または 4 を比較キーとして使用することが決定された場合、LiveCompare はキーの一意性もチェックします。一意性が不可能な場合、LiveCompare はその特定のテーブルでの比較を中止します（この動作は **check_uniqueness_enforcement = false** で無効にできます）。

2.5.3 修正する違い

LiveCompare は、次の違いを特定して修正を提供できます。

- 大部分のデータ接続に行が存在します。修正は、異なるデータベースの **INSERT** です。
- 大部分のデータ接続に行が存在しません。修正は、異なるデータベースの **DELETE** です。
- すべてのデータベースに行が存在しますが、一部の列の値が一致しません。修正は、異なるデータベースの **UPDATE** になります。

デフォルトでは **difference_statements = all**。これは、LiveCompare が見つけた違いごとに 3 つの DML タイプ (**INSERT**、**UPDATE**、および **DELETE**) をすべて適用しようとすることを意味します。ただし、設定 **difference_statements** の値を変更することにより、違いの修正を提供するときに検討する DML LiveCompare の種類を指定できます。

- **all** (デフォルト) : **INSERT s**、**UPDATE s**、および **DELETE s** を修正します。
- **inserts** : **INSERT s** のみを修正します。
- **updates** : **UPDATE s** のみを修正します。
- **deletes** : **DELETE s** のみを修正します。
- **inserts_updates** : **INSERT s** と **UPDATE s** のみを修正します。
- **inserts_deletes** : **INSERT s** と **DELETE s** のみを修正します。
- **updates_deletes** : **UPDATE s** と **DELETE s** のみを修正。

difference_statements の値が **all**、**updates**、**inserts_updates** または **updates_deletes** の場合、**NULL** を列に設定する **UPDATE** を無視するように LiveCompare に指示できます。

2.5.4 差分ログ

表 `difference_log` は、LiveCompare がそれらをチェックするたびに、違いに関するすべての情報を保存します。ユーザーは再確認モードで LiveCompare を複数回実行できるため、この表は、LiveCompare が再確認した時間枠で違いがどのように変化したかを示しています。

- **検出 (D)** : 違いが検出されました。再確認モードと修正モードでは、LiveCompare はすべての永久差とタイ差を検出済みとしてマークします。
- **永続的 (P)** : 違いを再確認した後、データがまだ発散している場合、LiveCompare は違いを**永続的**としてマークします。
- **同点 (T)** : Permanent と同じですが、どの接続が多数派であるかを判断するための十分なコンセンサスがありません。
- **Absent (A)** : 再チェック時に違いが存在しないことを LiveCompare が検出した場合（行は両方のデータベースで一貫しています）、LiveCompare は違いを **Absent** としてマークします。
- **揮発性 (V)** : 再チェック時に一貫性のない行で `xmin` が変更された場合、LiveCompare は違いを **揮発性** としてマークします。
- **無視 (I)** : ユーザーは、出力 PostgreSQL 接続で関数 `<livecompare_schema_name>.accept_divergence(session_id, table_name, difference_pk)` を手動で呼び出すことにより、特定の違いの再チェックを停止できます。例：

```
SELECT livecompare.accept_divergence(
    2                -- session_id
, public.categories -- table_name
, $(10)$(10)       -- difference_pk
);
```

2.6 BDR Support

LiveCompare は、BDRBDR に対しても使用できます。

`logical_replication_mode = bdr` を設定すると、ツールは比較されるすべてのデータベースが同じ BDR クラスターに属すると想定します。次に、接続としてノード名を指定し、テーブルをフィルタリングするためのレプリケーションセットを指定できます。

たとえば、BDR クラスター内の任意のノードに接続できるとします。これを **Initial Connection** と呼びましょう。このノードに最初に接続することにより、LiveCompare は BDR メタデータを確認し、他のすべてのノードから接続情報を取得できます。

次に、3 つの BDR ノードを比較するとします。LiveCompare は **Initial Connection** から始まる任意のノードに接続できるため、`dsn` やデータ接続の接続情報を定義する必要はありません。`node_name` を定義するだけです。LiveCompare は、そのノードの接続情報に関する BDR メタデータを検索し、ノードに接続します。

LiveCompare が BDR メタデータを取得して他のすべてのノードに接続できるようにするには、LiveCompare が BDR ビュー `bdr.node_summary`、フィールド `interface_connstr` から同じ DSN を使用してノードに接

続できる必要があることに注意してください。この場合、**Initial Connection** と同じマシンで **postgres** ユーザーとして **LiveCompare** を実行することをお勧めします。それが不可能な場合は、すべてのデータ接続で **dsn** 属性を定義してください。

レプリケーションセットをテーブルフィルターとして指定することもできます。**LiveCompare** は、**BDR** メタデータを使用して、**replication_sets** 設定で定義したレプリケーションセットに属するテーブルのみを考慮して、テーブルリストを構築します。

たとえば、**.ini** ファイルを作成して 3 つの **BDR** ノードを比較できます。

```
[General Settings]
logical_replication_mode = bdr
max_parallel_workers = 4

[Initial Connection]
dsn = port=5432 dbname=live user=postgres

[Node1 Connection]
node_name = node1

[Node2 Connection]
node_name = node2

[Node3 Connection]
node_name = node3

[Output Connection]
dsn = port=5432 dbname=liveoutput user=postgres

[Table Filter]
replication_sets = set_name = bdrgroup
```

BDR クラスター内のすべてのアクティブノードを比較するように **LiveCompare** に指示することもできます。そのためには、次のようにします。

- **General Settings** で、**all_bdr_nodes = on** を有効にします。
- **Initial Connection** を指定します。
- 追加のデータ接続は必要ありません。

例：

```
[General Settings]
logical_replication_mode = bdr
max_parallel_workers = 4
all_bdr_nodes = on

[Initial Connection]
```

(次のページに続く)

(前のページからの続き)

```
dsn = port=5432 dbname=live user=postgres
```

[Output Connection]

```
dsn = port=5432 dbname=liveoutput user=postgres
```

[Table Filter]

```
replication_sets = set_name = bdrgroup
```

`all_bdr_nodes = on` の場合、LiveCompare は `Initial Connection` を使用してすべての BDR ノードのリストを取得します。追加のデータ接続は必要ありません。設定されている場合、データ接続のリストに追加されます。たとえば、移行プロジェクトで役立つ、BDR クラスター全体を単一の Postgres 接続と比較できます。

[General Settings]

```
logical_replication_mode = bdr
```

```
max_parallel_workers = 4
```

```
all_bdr_nodes = on
```

[Initial Connection]

```
dsn = port=5432 dbname=live user=postgres
```

[Old Connection]

```
dsn = host=oldpg port=5432 dbname=live user=postgres
```

[Output Connection]

```
dsn = port=5432 dbname=liveoutput user=postgres
```

[Table Filter]

```
replication_sets = set_name = bdrgroup
```

設定 `node_name` および `replication_sets` は、次のテクノロジーでサポートされています。

- BDR 1、2、3 および 4;
- pglogical 2 および 3。

BDR の代わりに pglogical メタデータのフェッチを有効にするには、`logical_replication_mode = bdr` の代わりに `logical_replication_mode = pglogical` を設定するだけであることを注意してください。

2.6.1 BDR Witness ノード

BDR のレプリケーションセットを使用すると、BDR レプリケーションに含めるように特定のテーブルを構成し、テーブルが属するレプリケーションセットをサブスクライブするようにノードを構成することにより、そのようなテーブルからデータを受信するノードを指定できます。これにより、BDR シャーディングや BDR Witness ノードの使用など、さまざまなアーキテクチャが可能になります。

BDR Witness は、他のノードから DML をレプリケートしない通常の BDR ノードです。Witness の目的は、Raft コンセンサス投票でクォーラムを提供することです（BDR Witness ノードの詳細については、BDR のドキュメントを確認してください）。レプリケーションセットの構成に応じて、Witness は DDL をレプリケートする場合としない場合があります。これは、2 種類の BDR があることを意味します。

- データもテーブルもない完全に空のノード。または
- 他のノードから DDL をレプリケートするため、空のテーブルを持つノード。

最初のケースでは、BDR Witness が比較に含まれていても（[Connections] で手動で、または `all_bdr_nodes = on` を使用して）、Witness にはテーブルがないため、次のメッセージがログに記録されます。

```
Table public.tbl does not exist on connection node1
```

一方、2 番目のケースでは、テーブルは BDR Witness に存在します。ただし、Witness で欠落しているデータを相違として報告するのは正しくありません。そのため、各テーブルについて、LiveCompare は比較に含まれる各ノードに関する次の情報を確認します。

- ノードがサブスクライブするレプリケーションセット。
- テーブルが関連付けられているレプリケーションセット。
- レプリケーションは、Table Filter の下のフィルター `replication_sets` で定義されたユーザーを設定します。

BDR Witness の場合、レプリケーションセットの 3 つすべてのリストの共通部分が空の場合、LiveCompare はこれをログに記録します。

```
Table public.tbl is not subscribed on connection node1
```

どちらの場合も、その特定のテーブルの比較はテーブルが存在するノードで進行し、テーブルはレプリケーションセット構成に従って複製されます。

2.6.2 BDR クラスターの違い

LiveCompare はローカルノードのみに変更を加えます。修正変更が他のノードにレプリケートされないことが重要です。

`logical_replication_mode = bdr` の場合、LiveCompare は最初に `bdr_local_only_origin` という複製起点が既に存在するかどうかを確認します（複製起点の名前は、設定 `difference_fix_replication_origin` を調整することで構成できます）。`bdr_local_only_origin` というレプリケーションオリジンがまだ存在しない場合、LiveCompare はすべての BDR 接続でそれを作成します。

重要：BDR 3.6.18 では、ローカルのみトランザクションを適用するために使用される、事前に作成された新しい `bdr_local_only_origin` レプリケーションオリジンが導入されたことに注意してください。したがって、LiveCompare が BDR 3.6.18 に接続されている場合、このレプリケーションオリジンは作成されません。

LiveCompare は、次を考慮して適用スクリプトを生成します。

- レプリケーションオリジン `bdr_local_only_origin` を使用するように現在のトランザクションを設定します。したがって、実行される DML は `bdr_local_only_origin` に関連付けられます。
- 現在のトランザクションの日時をはるかに過去に設定するため、データベースで実行されている実際の DML との BDR 競合がある場合、LiveCompare DML は常に競合を失います。

LiveCompare 修正スクリプトを BDR ノードに適用した後、次のクエリを使用して、LiveCompare によって挿入または更新された行を正確に取得できます (`mytable` をテーブルの名前に置き換えます)。

```
with lc_origin as (
    select roident
    from pg_replication_origin
    where rname = bdr_local_only_origin
)
select t.*
from mytable t
inner join lc_origin r
on r.roident = bdr.pg_xact_origin(t.xmin);
```

(削除された行は表示されないことに注意してください。)

LiveCompare でメタデータを適切に取得するには、少なくとも `bdr_superuser` 権限を持つ PostgreSQL ユーザーが必要です。

PostgreSQL ユーザーに `bdr_superuser` または PostgreSQL スーパーユーザー権限がある場合、レプリケーション起点を含む上記のすべての手順は出力スクリプトにのみ適用されます。それ以外の場合、LiveCompare はレプリケーションオリジンを関連付けずに修正を生成します (トランザクションレプリケーションは `SET LOCAL bdr.xact_replication = off` を使用してまだ無効になっています)。ただし、DML スクリプトを適用するときはレプリケーションオリジンを使用することをお勧めします。そうしないと、LiveCompare は競合解決に関して通常のユーザーアプリケーションと同じ優先順位になります。また、修正に関連付けられた複製起点がないため、LiveCompare によって修正されたすべての行をリストする上記のクエリは使用できません。

BDR 3.6.18 と BDR 3.7.0 の間では、次の機能が使用されます。

- `bdr.difference_fix_origin_create()`：この複製起点が存在しない場合、LiveCompare によって実行され、`difference_fix_replication_origin` で指定された複製起点 (デフォルトでは `bdr_local_only_origin` に設定されます)。
- `bdr.difference_fix_session_setup()`：生成された DML スクリプトに含まれるため、トランザクションは `difference_fix_replication_origin` で指定された複製起点に関連付けられます。
- `bdr.difference_fix_xact_set_avoid_conflict()`：生成された DML スクリプトに含まれているため、トランザクションは過去に設定されます (2010-01-01) ため、LiveCompare によって適用される修正トランザクションは常に競合を失います。

上記の機能には、PostgreSQL スーパーユーザーではなく `bdr_superuser` が必要です。BDR 3.7.0 以降、これらの機能は非推奨です。LiveCompare は、PostgreSQL スーパーユーザーとして実行している場合、代わりに次の関数を使用して、上記と同じアクションを実行します。

- `pg_replication_origin_create(origin_name);`
- `pg_replication_origin_session_setup();`
- `pg_replication_origin_xact_setup()。`

PostgreSQL スーパーユーザーが使用されていない場合、LiveCompare は生成された DML トランザクションに次のみを含めます。

```
SET LOCAL bdr.xact_replication = off;
```

2.6.3 BDR の競合

LiveCompare には `conflicts` という実行モードがあります。この実行モードは、BDR クラスターに固有です。BDR 3.6、BDR 3.7、または BDR 4 クラスターでのみ機能します。

`compare` モードはテーブルのすべての内容を全体的に比較するために使用されますが、`conflicts` モードは BDR 3.6 の場合は `bdr.apply_log` に、BDR の場合は `bdr.conflict_history` に登録されている既存の競合に関連するタプル/テーブルにのみ焦点を合わせます 3.7 および BDR 4。

とはいえ、`conflicts` 実行モードは、特定のテーブルの特定のタプルを検査するだけなので、`compare` モードよりもはるかに高速に実行されることが期待されます。同時に、同じ理由で、`compare` モードほど完全ではありません。

この実行モードの主な目的は、BDR によって実行される自動競合解決がノード間で一貫していること、つまり、BDR が競合を解決した後、クラスターが一貫した状態にあることを確認することです。

一般的なユースケースでは、自動競合解決によりクラスターの一貫性が保証されますが、自動競合解決によりノード間でタプルが分散することが知られているケースがいくつかあります。したがって、LiveCompare の `conflicts` 実行モードは、時間と結果のバランスをとりながら、一貫性のチェックと保証に役立ちます。

競合の例

`node3` で次のクエリを実行するとします。

```
SELECT c.reloid::regclass,
       s.origin_name,
       c.local_time,
       c.key_tuple,
       c.local_tuple,
       c.remote_tuple,
       c.apply_tuple,
       c.conflict_type,
       c.conflict_resolution
```

(次のページに続く)

(前のページからの続き)

```
FROM bdr.conflict_history c
INNER JOIN bdr.subscription_summary s
ON s.sub_id = c.sub_id;
```

bdr.conflict_history には次の競合が見られます。

reloid		tbl
origin_name		node2
local_time		2021-05-13 19:17:43.239744+00
key_tuple		{"a":null,"b":3,"c":null}
local_tuple		
remote_tuple		
apply_tuple		
conflict_type		delete_missing
conflict_resolution		skip

これは、DELETE が node2 から node3 に到着したとき、テーブル tbl に b = 3 の行がなかったことを意味します。ただし、INSERT は node1 から node3 に後で到着した可能性があり、次に b = 3 を持つ行を node3 に追加しました。したがって、これは node3 の現在の状況です。

```
bdrdb=# SELECT * FROM tbl WHERE b = 3;
 a | b | c
-- +-----+-----
 x | 3 | foo
(1 row)
```

ノード node1 および node2 で、これが表示されます。

```
bdrdb=# SELECT * FROM tbl WHERE b = 3;
 a | b | c
-- +-----+-----
(0 rows)
```

BDR クラスターは発散しています。

このような相違を検出して修正するには、LiveCompare を compare モードで実行できますが、比較セットのサイズによっては（テーブル tbl が非常に大きいと想像してください）、時間がかかる場合があります。

これは、まさに conflicts モードが役立つ状況です。この場合、delete_missing の競合は node3 からのみ可視ですが、LiveCompare は競合がログに記録された行（key_tuple、local_tuple、remote_tuple および apply_tuple）から PK 値を抽出し、影響を受けるテーブルでのみクラスター全体の比較を実行できます。すでに PK 値でフィルタリングしています。次に、比較により、クラスター内のすべてのノードの現在の行バージョンが確認されます。

そこで、check.ini ファイルを作成して all_bdr_nodes = on を設定します、つまり、クラスター内のすべてのノードを比較するように LiveCompare に指示します。

```
[General Settings]
logical_replication_mode = bdr
max_parallel_workers = 2
all_bdr_nodes = on

[Initial Connection]
dsn = dbname=bdrdb

[Output Connection]
dsn = dbname=liveoutput
```

conflicts モードで LiveCompare を実行するには：

```
livecompare check.ini --conflicts
```

実行後、コンソール出力に次のように表示されます。

```
Elapsed time: 0:00:02.443557
Processed 1 conflicts about 1 tables from 3 connections using 2 workers.
Found 1 divergent conflicts in 1 tables.
Processed 1 rows in 1 tables from 3 connections using 2 workers.
Found 1 inconsistent rows in 1 tables.
```

フォルダー `./lc_session_X/`（現在の比較セッションの番号は `X`）の中に、LiveCompare はファイル `conflicts_DAY.out`（ファイル名の `DAY` を現在の日に置き換えます）を書き込み、すべての発散する競合に関する主な情報を表示します。

データベース `liveoutput` に接続すると、たとえば次のクエリを使用して、競合に関する詳細を表示できます。

```
SELECT *
FROM livecompare.vw_conflicts
WHERE session_id = 1
      AND conflict_id = 1
ORDER BY table_name,
         local_time,
         target_node;
```

次のように表示されます。

session_id	1
table_name	public.tbl
conflict_id	1
connection_id	node3
origin_node	node2
target_node	node3
local_time	2021-05-13 19:17:43.239744+00

(次のページに続く)

(前のページからの続き)

key_tuple	{"a": null, "b": 3, "c": null}
local_tuple	
remote_tuple	
apply_tuple	
conflict_type	delete_missing
conflict_resolution	skip
conflict_pk_value_list	{(3)}
difference_log_id_list	{1}
is_conflict_divergent	t

`is_conflict_divergent = true` は、LiveCompare が競合を比較し、競合によって報告されたテーブルと行でノードが現在発散していることを意味します。ビュー `livecompare.vw_conflicts` は、非発散のものを含むすべての競合に関する情報を表示します。

LiveCompare は、DML スクリプト `./lc_session_X/apply_on_the_node3_DAY.sql` も自動的に生成します（ファイル名の `DAY` を現在の日に置き換えます）。

```
BEGIN;

SET LOCAL bdr.xact_replication = off;
SELECT pg_replication_origin_session_setup(bdr_local_only_origin);
SELECT pg_replication_origin_xact_setup(0/0, 2010-01-01::timestampz);

SET LOCAL ROLE postgres;
DELETE FROM public.tbl WHERE (b) = (3);

COMMIT;
```

他の 2 つのノードには行が存在しないため、LiveCompare は `node3` から `b = 3` を `DELETE` することを提案しています。デフォルトでは、LiveCompare はノードの大部分に基づいて修正する DML を提案します。

`node3` に対してこの DML スクリプトを実行した場合:

```
psql -h node3 -f ./lc_session_X/apply_on_the_node3_DAY.sql
```

BDR クラスターの一貫性が再び得られます。

`--conflicts` モードの比較は完全な `--compare` よりもはるかに高速であるため、競合解決によってクラスター全体の一貫性が提供されるように、`--conflicts` 比較セッションをより頻繁にスケジュールすることを強くお勧めします。

BDR 3.7 の `bdr.conflict_history` または BDR 3.6 の `bdr.apply_log` のデータを表示できるようにするには、`bdr_superuser` または PostgreSQL スーパーユーザーで LiveCompare を実行する必要があることに注意してください。

競合フィルター

`bdr.conflicts_history` または `bdr.apply_log` のいずれかの列で競合をフィルター処理するように LiveCompare に指示することもできます。例：

```
[Conflicts Filter]
conflicts = table_name = public.tbl and conflict_type = delete_missing
```

2.6.4 混合技術

`node_name` と `replication_sets` のメタデータは **Initial Connection** で取得されることに注意してください。したがって、`pglogical` および/または `BDR` 対応データベースである必要があります。

テーブルのリストは、最初のデータ接続で作成されます。したがって、`replication_sets` 条件は最初の接続で有効である必要があります。

混合技術の比較を実行できます。例：

- BDR 1 ノード対 BDR 3 ノード。
- BDR 4 ノードとバニラの Postgres インスタンス;
- Vanilla Postgres インスタンスと `pglogical` ノード。

2.7 Oracle Support

LiveCompare を使用して、Oracle データベースのデータを任意の数の PostgreSQL または BDR データベースと比較できます。

たとえば、データ接続で `technology = oracle` を定義できます。他の設定を使用して、Oracle への接続を定義できます。

- `host`
- `port`
- `service`
- `user`
- `password`

他のすべてのデータ接続は PostgreSQL である必要があります。

Oracle データベースと PostgreSQL データベースを比較する簡単な例を次に示します。

```
[General Settings]
logical_replication_mode = off
max_parallel_workers = 4
oracle_user_tables_only = on
```

(次のページに続く)

(前のページからの続き)

```

oracle_ignore_unsortable = on
column_intersection = on
force_collate = C
difference_tie_breakers = oracle

[Oracle Connection]
technology = oracle
host = 127.0.0.1
port = 1521
service = XE
user = LIVE
password = live

[Postgres Connection]
technology = postgresql
dsn = dbname=liveoracle user=william

[Output Connection]
dsn = dbname=liveoutput user=william

[Table Filter]
schemas = schema_name = live

```

ここで、Oracle の `schema_name` はユーザーテーブルサンドボックスです。すべてのテーブル名はデフォルトでスキーマ修飾されます。

- Postgres : <schema_name> . <table_name>
- オラクル : <user> . <table_name>

`schema_qualified_table_names = off` を設定することで、スキーマ修飾テーブル名を無効にすることができます。これは、`oracle_user_tables_only = on` の場合にのみ実行できます。これは、接続されている Oracle ユーザーに属するテーブルのみを検索することを意味します。スキーマ修飾テーブル名が無効になっている場合、Postgres では、ロールまたは構成にデフォルトの `search_path` を設定する必要があります。または、接続 `start_query` パラメーターを使用して適切な `search_path` を設定します。

```

[General Settings]
logical_replication_mode = off
max_parallel_workers = 4
oracle_user_tables_only = on
oracle_ignore_unsortable = on
column_intersection = on
force_collate = C
difference_tie_breakers = oracle
schema_qualified_table_names = off

```

(次のページに続く)

(前のページからの続き)

```
[Oracle Connection]
technology = oracle
host = 127.0.0.1
port = 1521
service = XE
user = LIVE
password = live

[Postgres Connection]
technology = postgresql
dsn = dbname=liveoracle user=william
start_query = SET search_path = myschema1, myschema2, public

[Output Connection]
dsn = dbname=liveoutput user=william

[Table Filter]
tables = table_name in (mytable1, mytable2)
```

schema_qualified_table_names = off の場合、Table Filter、Row Filter、および Column Filter でも非修飾テーブル名を使用できます。

また、LiveCompare から進行状況とレポート情報を書き込むには Output Connection が必要であることに注意してください。

BDR データベースを Oracle と比較する必要がある、(前述のように) Initial Connection、node_name および replication_sets 機能を利用したい場合は、次のように、最後のデータ接続を Oracle を指すことができます。

```
[General Settings]
logical_replication_mode = bdr
max_parallel_workers = 4
oracle_user_tables_only = on
oracle_ignore_unsortable = on
column_intersection = on
force_collate = C
difference_tie_breakers = oracle

[Initial Connection]
dsn = port=5432 dbname=live user=postgres

[BDR Connection]
node_name = node1

[Oracle Connection]
technology = oracle
```

(次のページに続く)

(前のページからの続き)

```
host = 127.0.0.1
port = 1521
service = XE
user = LIVE
password = live

[Output Connection]
dsn = port=5432 dbname=liveoutput user=postgres

[Table Filter]
replication_sets = set_name = bdrgroup
```

次のように、BDR クラスタ全体を単一の Oracle データベースと比較することもできます。

```
[General Settings]
logical_replication_mode = bdr
max_parallel_workers = 4
oracle_user_tables_only = on
oracle_ignore_unsortable = on
column_intersection = on
force_collate = C
difference_tie_breakers = oracle
all_bdr_nodes = on

[Initial Connection]
dsn = port=5432 dbname=live user=postgres

[Oracle Connection]
technology = oracle
host = 127.0.0.1
port = 1521
service = XE
user = LIVE
password = live

[Output Connection]
dsn = port=5432 dbname=liveoutput user=postgres

[Table Filter]
replication_sets = set_name = bdrgroup
```

2.7.1 要件

LiveCompare は、追加のソフトウェアをインストールすることなく、すぐに PostgreSQL データベースで動作することに注意してください。

ただし、Oracle に接続できるようにするには、LiveCompare に追加のソフトウェアが必要です。

Oracle インスタントクライアント

Oracle Instant Client をダウンロードしてインストールする必要があります（または、使用しているオペレーティングシステムによっては特定のフォルダーに抽出します）。

- **MacOSX** : Oracle Instant Client **64-bit**) をダウンロードし、~/lib で抽出します。
- **Linux** : Oracle Instant Client **32-bit**) **64-bit**) をダウンロードしてシステムにインストールしてから、LD_LIBRARY_PATH ;
- **Windows** : Oracle Instant Client **32-bit**) **64-bit**) をダウンロードして、LiveCompare フォルダーに抽出します。

cx_Oracle Python モジュール

LiveCompare を Oracle Database に接続するには、Python モジュール

cx_Oracle Python モジュール をシステムにインストールして使用できるようにする必要があります。

現在、**cx_Oracle** は Linux ディストリビューションのリポジトリからインストールできないため、**the instructions in cx_Oracle website** に従ってシステムにインストールしてください。

LiveCompare は postgres オペレーティングシステムユーザーで実行することをお勧めします。次のコマンドを使用して、postgres ユーザーのみが PIP を介して **cx_Oracle** モジュールをインストールできます。

```
pip3 install --user cx_Oracle --upgrade
```

2.7.2 違い

LiveCompare が違いを見つけると、PostgreSQL 接続にのみ適用される DML スクリプトを生成します。Oracle 接続で適用する DML スクリプトは生成されません。

2.7.3 BLOB および CLOB データ型

LiveCompare は、PostgreSQL の同等のフィールドが `text` 型である場合、Oracle の CLOB フィールドを比較できます。Oracle の BLOB フィールドについても同じことが言えます。PostgreSQL での同等のフィールドは `bytea` 型である必要があります。

ただし、デフォルトでは、LiveCompare は BLOB / CLOB フィールドが主キーにある場合、またはテーブルに主キーがない場合、処理しません。その場合、テーブルは無視され、LiveCompare ログに次のようなメッセージが表示されます。

```
ORA-00932: inconsistent datatypes: expected - got BLOB
```

General Settings セクションで次の 2 つの設定を有効にすることにより、テーブルに主キーがない場合に BLOB/CLOB フィールドを無視するように LiveCompare に指示することで、この動作を回避できます。

```
oracle_ignore_unsortable = on
column_intersection = on
```

2.7.4 互換性のない照合

Oracle では、一般に次の初期化パラメーターが設定されています。

```
NLS_COMP = BINARY
NLS_SORT = BINARY
```

これは、NLS_LANG および他の言語設定に関係なく、Oracle のすべての ORDER BY 操作が文字バイナリコードを使用して実行されることを意味します。

Postgres では、同じ動作を示す同等の照合は C 照合です。Postgres データベースが別の照合順序で初期化された場合、デフォルトで LiveCompare は PK 値を並べ替えるときに問題を検出し、誤検出につながる可能性があります。

それを回避するには、Postgres で照合（たとえば、C 照合）を強制できるため、Oracle の同じソート動作と一致します。

```
force_collate = C
```

比較セッションに Oracle と PostgreSQL が含まれることを LiveCompare が検出した場合、ユーザーが他の値に設定していない限り、LiveCompare は既に `force_collate = C` を設定しています。

2.7.5 共通ハッシュ

デフォルトでは、PostgreSQL と Oracle を比較した場合でも、LiveCompare には `comparison_algorithm = block_hash` があります。ただし、「共通ハッシュ」は次のルールに従って構築されます。

- すべての列値を連結することにより、行は完全にテキストとして表されます。
- Postgres 側では、タイムスタンプ、数値、および bytea データ型が処理されて Oracle を模倣します。
- この方法では、完全な行表現が両側で MD5 を使用してハッシュされます。
- これにより、`comparison_algorithm` を `block_hash` および `row_hash` に設定できます。
- `block_hash` の使用時に不一致がある場合、LiveCompare は Postgres と Postgres の比較の場合と同様に、自動的に `row_hash`、次に `full_row` にフォールバックします。
- Oracle の BLOB、CLOB、および NCLOB フィールドは最初の 2000 文字のみに制限されています (`comparison_algorithm = full_row` では BLOB と CLOB 全体を比較できます)。
- Oracle では、完全な行表現の幅は 4000 文字を超えてはなりません。

完全な行表現が 4000 文字を超える場合、LiveCompare はその特定のテーブルの比較を中止し、ログに次のエラーメッセージが表示されます。

```
ORA-01489: result of string concatenation is too long
```

さらなる LiveCompare バージョンは、これらの特定のテーブルで `full_row` 比較にフォールバックします。当面の回避策は、`comparison_algorithm = full_row` を使用して、これらのテーブルのみに別個の比較セッションを構成することです。

共通ハッシュは、Oracle 12c 以降で `standard_hash` ファンクションを使用します。Oracle 11g では、`standard_hash` ファンクションを使用できないため、LiveCompare は代わりに `dbms_crypto.hash` ファンクションを使用しようとしていますが、たとえば、Oracle 側のユーザーに追加の権限が必要になる場合があります。

```
GRANT EXECUTE ON sys.dbms_crypto TO testuser;
```

2.8 Settings

2.8.1 一般設定

- `logical_replication_mode` : プログラムが接続とテーブルフィルター設定を解釈する方法 (詳細は以下を参照)、および比較を開始する前に接続で確認する要件に影響します。現在、可能な値は次のとおりです。

```
- `off`: Assumes there is no logical replication between the databases;

- `native`: Assumes there is native logical replication between the
databases. Enables the usage of the `Table Filter -> publications`
```

(次のページに続く)

(前のページからの続き)

setting to specify the list of tables to be used. Requires PostgreSQL 10+ on all databases.

- ``pglogical``: Assumes there is pglogical replication between the databases. Enables the usage of the ``Table Filter -> replication_sets`` setting to specify the list of tables to be used. Also enables the usage of ``node_name`` to specify the data connections, which require setting the ``Initial Connection`` that is used to retrieve DSN information of the nodes. Requires the ``pglogical`` extensions to be installed on all databases.

- ``bdr``: Assumes all data connections are nodes from the same BDR cluster. Enables usage of ``Table Filter -> replication_sets`` setting to specify list of tables to be used. Also enables usage of ``node_name`` to specify the data connections, which require setting ``Initial Connection`` that is used to retrieve DSN information of the nodes. Requires ``pglogical`` and ``bdr`` extensions installed on all databases.

- `all_bdr_nodes` : `logical_replication_mode` が `bdr` に設定されている場合、初期接続（以下を参照）のみを指定し、LiveCompare にアクティブな BDR ノードの現在のリストに基づいて接続リストを構築させることができます。デフォルト：off。

• `max_parallel_workers` : 検討する並列プロセス数。各プロセスは、キューのテーブルで動作します。デフォルト：2。

重要 : 各プロセスは $N + 1$ 個の開いた接続を維持します。1 つは各データ接続に、もう 1 つは出力データベースに。

- `buffer_size` : データフェッチ操作ごとにテーブルから取得される行数。デフォルト：4096。
- `log_level` : ログファイルの詳細レベル。可能な値：debug、info、warning または error。デフォルト：info。
- `data_fetch_mode` : LiveCompare がデータベースからデータを取得する方法に影響します。
- `prepared_statements` : データフェッチにプリペアドステートメント（LIMIT を使用したクエリ）を使用します。毎回非常に少量のデータ（デフォルトでは `buffer_size = 4096` 行）のみがフェッチされるため、3 つのモードすべての中で最も影響が少なく、同じ理由でより安全なフェッチモードです。非同期データフェッチを許可します（`parallel_data_fetch` で定義）。一般的なユースケースでは、このフェッチ方法は優れたパフォーマンスを提供しますが、大きなテーブルではパフォーマンスの低下が感じられる場合があります。これはデフォルトであり、サーバーの負荷が中程度の場合に強くお勧めします。
- `server_side_cursors_with_hold` : データフェッチにサーバー側カーソル WITH HOLD を使用します。テーブルデータは単一のトランザクションで取得されるため、xmin が抑制され、膨張とレプリケーションの問題が発生する可能性があります。また、VACUUM が正常に実行されません。また、WITH HOLD 句は、クエリをマテリアライズするように Postgres に指示します（ワーカーはデータがマテリアライズされるまで数秒間ハングする場合があります）。その影響はすべて `parallel_chunk_rows`（デフォルトでは無効）を使用することで軽減でき、`buffer_size` を少し増やすと速度が向上します。非同期データフェッチを許可します（`parallel_data_fetch` で定義）。一般的なユースケース

では、`prepared_statements` と比較した場合、このフェッチ方法には利点はありませんが、複数の小さなテーブルの場合は高速です。ただし、このモードは、テストや移行シナリオなど、負荷が非常に低い場合にのみお勧めします。

- `server_side_cursors_without_hold` : データフェッチにサーバー側カーソル `WITHOUT HOLD` を使用します。`server_side_cursors_with_hold` と同様に、このモードは `xmin` を抑えることもできるため、Postgres で膨張、`VACUUM`、およびレプリケーションの問題が発生する可能性があります。 `WITHOUT HOLD` カーソルは比較セッション全体で開いているトランザクションを必要とするため、そのような影響は大きくなります（これは将来のバージョンで解除されます）。スナップショットは比較セッション全体にわたって保持されるため、ユースケースによっては比較結果が役立つ場合があります。クエリが具体化されないため、メモリ使用量と一時ファイルの生成は低いまです。非同期データの取得は許可されていません。パフォーマンスの面では、このモードは一般的なユースケースでは遅くなりますが、大きなテーブルの場合は速くなります。データベースの負荷が中程度の場合にお勧めします。

重要 : 適切なシナリオに適切な `data_fetch_mode` を選択することは非常に重要です。準備されたステートメントを使用すると、データベースサーバー上のフットプリントが最小になるため、最も安全なアプローチであり、一般的なユースケースに適しています。もう 1 つのポイントは、準備されたステートメントを使用すると、LiveCompare が常に最新バージョンの行を表示できることです。したがって、実稼働の高負荷サーバーには `prepared_statements` を使用することをお勧めします。テスト、移行シナリオ、および低負荷サーバーのいずれかの `server_side_cursors_*` 設定。最良の戦略は、おそらく非常に大きなテーブルには `server_side_cursors_without_hold` を、残りのテーブルには `prepared_statements` を混在させることです。費用対効果の比較については、以下の表を参照してください。

	<code>pre-prepared_statements</code>	<code>server_side_cursors_with_hold</code>	<code>server_side_cursors_without_hold</code>
<code>xmin hold</code>	very low	medium	high
<code>xmin released per</code>	buffer	chunk	whole comparison session
<code>temp files</code>	very low	very high	low
<code>memory</code>	very low	high	low
<code>allows async conns</code>	yes	yes	no
<code>fastest for</code>	general	small tables	large tables
<code>recommended load</code>	high	very low	low-medium

Oracle に関する注意 : Oracle の場合、`data_fetch_mode` 設定は完全に無視され、データは常に直接クエリを使用して Oracle から取得され、データはクライアント側のカーソルを介して `buffer_size` のチャンクで取得されます。

- `parallel_chunk_rows` : 並列比較のためにテーブルを複数のチャンクに分割することを検討するために必要な行の数。データの取得にはハッシュが使用されるため、ワーカーは互いに衝突しません。各テーブルのチャンクには、`parallel_chunk_rows` 以下の行があります。任意の値に設定する `< 1` はテーブル分割を無効にします。デフォルト : 0 (無効)。

重要 : テーブル分割は、複数のワーカーで大きなテーブルを並列で比較するのに役立ちますが、各ワーカーのパフォーマンスは、すべての行に適用されるハッシュ条件の影響を受ける可能性があります。Postgres の構成（特にデフォルトの `random_page_cost = 4`、これは最新のハードドライブには保守的すぎると考えられる）によっては、Postgres クエリプランナーが誤ってビットマップヒープスキャンを優先する場合があります。データベースが SSD で実行されている場合、LiveCompare を使用すると、比較のパフォーマンス

スを大幅に向上できます。これは、`start_query` 設定を使用して接続ごとに実行できます。

```
start_query = set enable_bitmapscan = off
```

- `parallel_data_fetch` : データフェッチを並列で実行する必要がある場合 (つまり、データベースへの非同期接続を使用します)。多元比較のパフォーマンスが向上しました。PostgreSQL 以外のデータ接続がある場合、この設定は自動的に無効になります。 `data_fetch_mode = prepared_statements` または `data_fetch_mode = server_side_cursors_with_hold` の場合にのみ許可されます。デフォルト: `on`。
- `comparison_algorithm` : LiveCompare がテーブルの行を介してデータを比較する方法に影響します。ハッシュの使用は、行全体の比較よりも高速です。次のいずれかの値をとります。

- ``full_row``: Disables row comparison using hashes. Full comparison, in this case, is performed by comparing the row column by column.

- ``row_hash``: Enables row comparison using hashes and enables table splitting. Tables are split so each worker compares a maximum of ``parallel_chunk_rows`` per table. Data row is hashed in PostgreSQL, so the comparison is faster than ``full_row``. However, if for a specific row the hash does not match, then for that specific row, LiveCompare will fallback to ``full_row`` algorithm (i.e., compare row by row). If any data connections is not PostgreSQL, then LiveCompare uses a row hash thats defined as the MD5 hash of the concatenated column values of the row, being considered a "common hash" among the database technologies being compared.

- ``block_hash``: Works the same as ``row_hash``, but instead of comparing row by row, LiveCompare builds a "block hash", i.e., a hash of the hashes of all rows in the data buffer that was just fetched (maximum of ``buffer_size`` rows). Conceptually it works like a 2-level Merkle Tree. If the block hash matches, then LiveCompare advances the whole block (this is why this comparison algorithm is faster than ``row_hash``). If block hash does not match, then LiveCompare falls back to ``row_hash`` and performs comparison row by row in the buffer to find the divergent rows. This is the default value.

- `min_time_between_heart_beats` : 「ハートビート」メッセージをログに記録するまで待機する時間 (秒)。各ワーカーは、比較されるラウンドパーツごとに個別に追跡します。デフォルト: 30 秒。
- `min_time_between_round_saves` : 比較アルゴリズムの進行中に各ラウンドの状態を更新するまで待機する時間 (秒)。ラウンドセーブはハートビート中にのみ発生するため、`min_time_between_round_saves` は `min_time_between_heart_beats` 以上である必要があります。ラウンドが終了すると、LiveCompare は常にそのテーブルのラウンド状態を更新することに注意してください。デフォルト: 60 秒。

重要 : ユーザーが `Ctrl-c` を押して LiveCompare の実行をキャンセルして再度開始すると、LiveCompare はラウンドの状態が保存されたポイントからそのテーブルのラウンドを再開します。

- `comparison_cost_limit` : `> 0` の場合、各ワーカーが `comparison_cost_delay` 秒の昼寝をする前に処理する行数に対応します。デフォルトは `0` で、各ワーカーは昼寝をせずに行を処理します。

- **comparison_cost_delay** : **comparison_cost_limit** > 0 の場合、この設定は各ワーカーがスリープする時間を指定します。デフォルトは 0.0 です。
- **stop_after_time** : ユーザーが Ctrl-c を押したかのように LiveCompare が自動的に停止するまでの秒単位の時間。中断された比較セッションがまだ終了していない場合、コマンドラインでセッション ID を引数として渡すことにより、再び再開できます。デフォルトは **stop_after_time** = 0 で、自動割り込みが無効になっていることを意味します。
- **consensus_mode** : どのデータ接続が発散しているかを判断するために LiveCompare が使用するコンセンサスアルゴリズム。可能な値は、**simple_majority**、**quorum_based** または **source_of_truth** です。**consensus_mode** = **source_of_truth** の場合、**difference_sources_of_truth** を入力する必要があります。デフォルトは **simple_majority** です。
- **difference_required_quorum** : **consensus_mode** = **quorum_based** の場合、この設定は、どの接続が発散するかを決定するために必要な最小クォーラムを指定します。0.0~1.0 の数値である必要があります (0.0 は接続が必要ないことを意味し、1.0 はすべての接続が必要であることを意味します。どちらの場合も極端なため使用しないでください)。デフォルト値は 0.5 で、それに近い値を使用することをお勧めします。
- **difference_sources_of_truth** : 信頼できるソースと見なされる接続名 (または **logical_replication_mode** = **bdr** および **all_bdr_nodes** = **on** の場合はノード名) のコンマ区切りリスト。**consensus_mode** = **source_of_truth** の場合にのみ使用されます。例: **difference_sources_of_truth** = **node1,node2**。この例では、セクション **node1 Connection** および **node2 Connection** を .ini ファイルで定義するか、**all_bdr_nodes** = **on** で **Initial Connection** のみを定義する必要がありますが、**node1** および **node2** は有効な BDR ノード名である必要があります。
- **difference_tie_breakers** : コンセンサスアルゴリズムがタイの状態を見つけるたびにタイブレーカーと見なされる接続名 (または **logical_replication_mode** = **bdr** および **all_bdr_nodes** = **on** の場合はノード名) のコンマ区切りリスト。例: **difference_tie_breakers** = **node1,node2**。この例では、セクション **node1 Connection** および **node2 Connections** を .ini ファイルで定義するか、**all_bdr_nodes** = **on** で **Initial Connection** のみを定義する必要がありますが、**node1** および **node2** は有効な BDR ノード名である必要があります。デフォルトでは、接続はタイブレーカーと見なされません。
- **difference_statements** : LiveCompare によって生成される DML ステートメントの種類を制御します。**difference_statements** の値は次のいずれかです。

```
- `all` (default)
- `inserts`
- `updates`
- `deletes`
- `inserts_updates`
- `inserts_deletes`
- `updates_deletes`
```

- **difference_allow_null_updates** : **UPDATE SET col = NULL** などのコマンドを差分レポートで許可するかどうかを決定します。デフォルト: **on**。
- **difference_statement_order** : LiveCompare によって生成される DML ステートメントの順序を

制御します。difference_statement_order の値は次のいずれかです。

```
- `delete_insert_update`
- `delete_update_insert` (default)
- `insert_update_delete`
- `insert_delete_update`
- `update_insert_delete`
- `update_delete_insert`
```

- **difference_fix_replication_origin** : BDR データベースを使用する場合、違いのために LiveCompare は特定のレプリケーション起点がまだ存在しない場合は作成し、そのレプリケーション起点を使用して DML 修正を含む適用スクリプトを作成します。設定 difference_fix_replication_origin は、LiveCompare が使用する複製起点の名前を指定します。ユーザーがこの設定に値を設定しない場合、LiveCompare は自動的に difference_fix_replication_origin = bdr_local_only_origin を設定します。LiveCompare が作成するレプリケーションオリジンは、比較後の検証を許可するために削除されないことに注意してください、必要に応じて、レプリケーションオリジンを後で手動で削除できます。logical_replication_mode = bdr が必要です。

重要 : BDR 3.6.18 では、ローカルのみトランザクションを適用するために使用される、事前に作成された新しい bdr_local_only_origin レプリケーションオリジンが導入されたことに注意してください。したがって、LiveCompare が BDR 3.6.18 に接続されている場合、このレプリケーションオリジンは作成されないため、ユーザーはこのレプリケーションオリジンを削除しないことをお勧めします。

- **difference_fix_start_query** : LiveCompare で生成された適用スクリプトの先頭で実行される任意のクエリ。さらに、BDR 比較が実行されており、difference_fix_start_query が空の場合、LiveCompare は自動的に次のことも行います。

```
- If the divergent connection is BDR 3.6.7, add
`SET LOCAL bdr.xact_replication = off;`
- Add commands that setup transaction to use the replication origin
specified in `difference_fix_replication_origin`.
```

- **show_progressBars** : コンソール出力に進行状況バーを表示するかどうかを決定します。この設定を無効にすると、バッチ実行に役立つ場合があります。デフォルト : on 。
- **output_schema** : 出力接続では、比較レポートテーブルが作成されます。デフォルト : livecompare 。
- **hash_column_name** : すべてのデータフェッチには、行内のすべての実際の列のハッシュである特定の列が含まれます。この設定は、この列の名前を指定します。デフォルト : livecompare_hash 。
- **rownumber_column_name** : 一部のフェッチでは、クエリ列内で row_number() 関数値を使用する必要があります。この設定は、この列の名前を指定します。デフォルト : livecompare_rownumber 。
- **fetch_row_origin** : この設定を有効にすると、LiveCompare は発散する各行のオリジン名を取得します。これはデバッグに役立つ場合があります。デフォルト : off 。
- **column_intersection** : この設定を有効にすると、比較されている特定のテーブルに対して、LiveCompare はすべての接続のテーブルの列の交差でのみ機能し、接続に存在する可能性のある追加の

列を無視します。この設定を無効にすると、LiveCompare はすべての接続のテーブルで列が同等であるかどうかを確認し、列が一致しない場合はテーブルの比較を中止します。デフォルト：off。

重要：テーブルに PK がある場合、`column_intersection = on` であっても、PK 列は異なることはできません。

- `ignore_nullable`：特定のテーブル比較について、LiveCompare が主キーとは異なる比較キーを使用している場合、`ignore_nullable` が有効になっている場合（デフォルト）、LiveCompare はすべての列を NOT NULL にする必要があります。 `ignore_nullable = off` を設定することで、その動作をオーバーライドできます。これにより、LiveCompare は比較で null 許容列を考慮します。一部のコーナーケースで誤検知が発生する可能性があります。
- `check_uniqueness_enforcement`：LiveCompare がユーザー定義の比較キーを使用しているか、テーブル内のすべての列を比較キーとして使用している場合、`check_uniqueness_enforcement` の設定が有効になっている場合（デフォルト）、LiveCompare は比較キーでテーブルの一意性をチェックします。
- `oracle_ignore_unsortable`：有効にすると、列がテーブル PK の一部でない場合、Oracle のソートできないデータ型（BLOB、CLOB、NCLOB、BFILE）の列を無視するように LiveCompare に指示します。この設定を有効にする場合は、`column_intersection` も有効にすることをお勧めします。
- `oracle_user_tables_only`：有効にすると、Oracle にログインしているユーザーからのみテーブルメタデータを取得するように LiveCompare に指示します。たとえば、`sys.all_tables` および `sys.all_tab_columns` の代わりに `sys.user_tables` および `sys.user_tab_columns` から読み取るため、より速くなります。デフォルト：off。
- `oracle_fetch_fk_metadata`：有効にすると、LiveCompare に外部キーのメタデータを取得するように指示します。これは処理が遅くなる場合があります。Oracle 接続の `fetch_fk_metadata` 設定の値をオーバーライドします。デフォルト：off。
- `schema_qualified_table_names`：本設定を有効にすると、テーブル名がスキーマ修飾されます。それを無効にすると、スキーマ修飾テーブル名を使用せずにテーブルを比較できます。Oracle x Postgres 比較では、`oracle_user_tables_only` も有効にする必要がありますが、Postgres x Postgres では、同じデータベース内であっても、異なるスキーマの下にあるテーブルを比較できます。また、`schema_qualified_table_names` が有効になっている場合、Table Filter -> tables、Row Filter、および Column Filter はスキーマ名なしでテーブル名を許可します。デフォルト：on。
- `force_collate`：off 以外の値かつ有効な照合名に設定されている場合、比較されるすべての Postgres データベースの ORDER BY 操作で指定された照合名を強制します。照合が異なる Postgres データベースを比較する場合、または Oracle と Postgres データベースを比較する場合に役立ちます（この場合、ユーザーは `force_collate = C` を設定する必要があります）。混合テクノロジー（Oracle と PostgreSQL など）を比較し、照合が指定されていない場合、値 C を想定します。デフォルト：off。
- `work_directory`：LiveCompare 作業ディレクトリへのパス。出力ファイルを含むセッションフォルダーがそのようなディレクトリに作成されます。デフォルト：.（現在のディレクトリ）。
- `abort_on_setup_error`：有効にすると、テーブル比較ラウンドを設定しようとしたときに LiveCompare でエラーが発生すると、比較セッション全体が中止されます。デフォルト：off。

重要：`abort_on_setup_error` の設定は、`compare` モードでのみ考慮されます。`recheck` モードでは、セットアップの最初のエラーで LiveCompare が常に中断します。

- `custom_dollar_quoting_delimiter` : LiveCompare が違いを見つけると、文字列にドル引用符を使用して DML を出力します。デフォルトの動作は、ランダムな文字列を作成して構成します。どうしてもカスタムのものを使用したい場合は、このパラメーターを使用する区切り文字として設定できます。定数の周りの \$ シンボルではなく、定数を設定するだけです。デフォルトは `off` です。これは、LiveCompare が単語 `LiveCompare` の md5 ハッシュを使用することを意味します。
- `session_replication_role_replica` : 有効にすると、LiveCompare は出力適用スクリプトで `session_replication_role` PostgreSQL 設定を `replica` として使用します。これは、発散のあるノードに DML を適用しているときにトリガーとルールが起動されないようにする場合に役立ちます。有効にするには PostgreSQL スーパーユーザが必要です。それ以外の場合は効果がありません。デフォルトは `off` です。
- `split_updates` : 有効にすると、LiveCompare は UPDATE の相違を分割します。つまり、UPDATE DML を生成する代わりに、適用スクリプトに対応する DELETE および INSERT を生成します。デフォルトは `off` です。
- `float_point_round` : データベースからの浮動小数点値を比較するときに LiveCompare が四捨五入する 10 進数を指定する整数。デフォルトは -1 で、浮動小数点の丸めが無効になります。

2.8.2 初期接続

初期接続は、`logical_replication_mode` が `pglogical` または `bdr` に設定されている場合にのみ使用され、ユーザーが `node_name` 設定のみを使用してデータ接続を設定している場合、プログラムの起動時にノード名から DSN を取得します。

- `technology` : RDBMS 技術。現在、使用可能な値は `postgresql` のみです。
- `dsn` : PostgreSQL 接続文字列。dsn が設定されている場合、`host`、`port`、`dbname` および `user` は無視されます。dsn 設定には、他のすべての `parameter key words allowed by libpq` を含めることもできます。
- `host` : サーバーアドレス。Unix ソケット接続を使用するには、空のままにします。
- `port` : ポート。デフォルト : 5432。
- `dbname` : データベース名。デフォルト : `postgres`。
- `user` : データベース利用者。デフォルト : `postgres`。
- `application_name` 。アプリケーション名。ユーザーが他のすべての接続情報の代わりに `dsn` を設定した場合でも使用できます。デフォルト : `livecompare_initial`。

2.8.3 出力接続

出力接続は、LiveCompare が比較レポートテーブルを作成する場所を指定します。

- **technology** : RDBMS 技術。現在、使用可能な値は `postgresql` のみです。
- **dsn** : PostgreSQL 接続文字列。dsn が設定されている場合、`host`、`port`、`dbname` および `user` は無視されます。dsn 設定には、他のすべての `parameter key words allowed by libpq` も含めることができます。
- **host** : サーバーアドレス。Unix ソケット接続を使用するには、空のままにします。
- **port** : ポート。デフォルト : 5432。
- **dbname** : データベース名。デフォルト : `postgres`。
- **user** : データベース利用者。デフォルト : `postgres`。
- **application_name**。アプリケーション名。ユーザーが他のすべての接続情報の代わりに `dsn` を設定した場合でも使用できます。デフォルト : `livecompare_output`。

2.8.4 データ接続

「データ接続」は `Initial Connection` および `Output Connection` に似た接続セクションですが、LiveCompare はデータ接続上のデータを効果的にフェッチして比較します。

`Initial Connection` や `Output Connection` と同様に、named section に「データコネクション」を定義します。セクション名は `Name Connection` の形式である必要があります。`Name` は、英字で始まる 1 語の文字列です。この場合、ユーザーが `Name` に入力するものは、データ接続の「接続 ID」と呼ばれます。また、各データ接続には、データ接続のリスト全体で一意的な接続 ID が必要です。

`logical_replication_mode = bdr` および `all_bdr_nodes = on` の場合、LiveCompare は `Initial Connection` から BDR メタデータを取得してデータ接続リストを構築するため、ユーザーはデータ接続を指定する必要はありません。

- **technology** : RDBMS 技術。現在可能な値は `postgresql` または `oracle` です。
- **node_name** : クラスター内のノードの名前。`logical_replication_mode` を `pglogical` または `bdr` に設定する必要があります。また、`Initial Connection` が満たされている必要があります。`node_name` が設定されている場合、`dsn`、`host port`、`dbname` および `user` の設定はすべて無視されます。
- **dsn** : PostgreSQL 接続文字列。dsn が設定されている場合、`host`、`port`、`dbname` および `user` は無視されます。dsn 設定には、他のすべての `parameter key words allowed by libpq` を含めることもできます。
- **host** : サーバーアドレス。Unix ソケット接続を使用するには、空のままにします。
- **port** : ポート。デフォルト : 5432。
- **dbname** : データベース名。デフォルト : `postgres`。
- **service** : Oracle 接続で使用するサービス名。デフォルト : `XE`。
- **user** : データベース利用者。デフォルト : `postgres`。

- **password** : 平文のパスワード。これを使用しないことをお勧めしますが、一部のレガシー接続では必要になる場合があります。
- **application_name** 。アプリケーション名。ユーザーが他のすべての接続情報の代わりに **dsn** または **node_name** を設定した場合でも使用できます。デフォルト: **livecompare_<Connection ID>**。
- **start_query** : データベースへの接続が開かれるたびに実行される任意のクエリ。
- **fetch_fk_metadata** : LiveCompare が接続上の外部キーに関するメタデータを収集する必要がある場合。デフォルト: **on**。

2.8.5 テーブルフィルター

省略するか空の場合、**.ini** ファイルのこのセクションは、**first** データベース内の **すべての** テーブルに対して LiveCompare を実行する必要があることを意味します。

特定のテーブルセットに対して LiveCompare を実行する場合、これを指定するさまざまな方法があります。

- **publications** : 特定のパブリケーションをフィルタリングでき、LiveCompare はそれらのパブリケーションに関連付けられたテーブルのみを使用します。変数 **publication_name** を使用して、次のように条件式を作成できます。

```
publications = publication_name = livepub
```

logical_replication_mode = native が必要です。

- **replication_sets** : **pglogical** または **BDR** を使用する場合、特定の複製セットをフィルタリングでき、LiveCompare はこれらの複製セットに関連付けられたテーブルでのみ機能します。変数 **set_name** を使用して、次のように条件式を作成できます。

```
replication_sets = set_name in (default, bdrgroup)
```

logical_replication_mode = pglogical または **logical_replication_mode = bdr** が必要です。

- **schemas** : 特定のスキーマをフィルタリングでき、LiveCompare はそれらのスキーマに属するテーブルでのみ機能します。変数 **schema_name** を使用して、次のように条件式を作成できます。

```
schemas = schema_name != badschema
```

- **tables** : 変数 **table_name** は、LiveCompare で動作させるテーブルのみをフィルタリングする条件式を作成するのに役立ちます。

```
tables = table_name not like %account
```

条件式では、**%** 文字を **%%** としてエスケープする必要があることに注意してください。

schema_qualified_table_names が無効になっていない限り、テーブル名はスキーマ修飾する必要があります。たとえば、特定のテーブルのリストのみをフィルター処理できます。

```
tables = table_name in (myschema1.mytable1, myschema2.mytable2)
```

一般的な設定 `schema_qualified_table_names` を無効にしている場合は、接続の `start_query` 設定で Postgres に適切な `search_path` も設定する必要があります。

```
[General Setting]
...
schema_qualified_table_names = off

[My Connection]
...
start_query = SET search_path TO myschema1, myschema2

[Table Filter]
tables = table_name in (mytable1, mytable2)
```

重要： `search_path` に設定された 2 つ以上のスキーマに同じ名前のテーブルが含まれている場合、最初に見つかったものが比較で考慮されることに注意してください。

Table Filter セクションには、`publications`、`replication_sets`、`schemas`、および `tables` フィルターを混在させることができ、LiveCompare は指定したすべてのフィルターの共通部分にあるテーブルのセットを検討します。例：

```
[Table Filter]
publications = publication_name = livepub
replication_sets = set_name in (default, bdrgroup)
schemas = schema_name != badschema
tables = table_name not like %account
```

また、テーブルリストを作成するために、テーブルフィルターが最初のデータベースに適用されることに注意してください。テーブルが最初のデータベースに存在し、フィルターで検討されているが、他のデータベースに存在しない場合、ログに次のように表示され、その特定のテーブルの比較はスキップされます。

```
2019-06-17 11:52:41,403 - ERROR - live_table.py - 55 - GetMetaData - P1: livecompare_
→second_1: Table public.test does not exist
2019-06-17 11:52:41,410 - ERROR - live_round.py - 201 - Initialize - P1: Table public.
→test does not exist on second connection. Aborting comparison
```

同様に、テーブルが他のデータベースに存在し、最初のデータベースに存在しない場合、テーブルフィルターを適用していなくても、比較で考慮されません。

テーブルの列名が完全に同じでない場合（`column_intersection` が有効になっている場合を除く）、特定のテーブルの比較もスキップされます。適切なメッセージもログファイルに含まれます。

現在、LiveCompare は、データ型と制約が両方のテーブルで同じかどうかを確認しません。

重要： `conflicts` モードはテーブルフィルターを利用しないことに注意してください。

2.8.6 行フィルター

このセクションでは、行レベルのフィルターを任意のテーブルに適用できるため、LiveCompare は行フィルターを満たす行でのみ機能します。

このセクションの下に、1 行に 1 つのテーブルのリストを書くことができます (schema_qualified_table_names が無効になっていない限り、すべてのテーブル名はスキーマ修飾する必要があります)。

[Row Filter]

```
public.table1 = id = 10
public.table2 = logdate >= 2000-01-01
```

この場合、テーブル public.table1 の場合、LiveCompare は句 id = 10 を満たす行でのみ機能しますが、テーブル public.table2 の場合、logdate >= '2000-01-01' を満たす行のみが比較で考慮されます。

一般的な設定 schema_qualified_table_names を無効にしている場合は、接続の start_query 設定で Postgres に適切な search_path も設定する必要があります。

[General Setting]

```
...
schema_qualified_table_names = off
```

[My Connection]

```
...
start_query = SET search_path TO public
```

[Row Filter]

```
table1 = id = 10
table2 = logdate >= 2000-01-01
```

任意の種類の SQL 条件 (WHERE 句に入れるのと同じ) は、テーブル行フィルターとして同じ行で受け入れられます。たとえば、大きなテーブルがあり、特定の数の ID のみを比較する場合、すべての ID で一時テーブルを作成できます。次に、IN 句を使用して、次のように JOIN をエミュレートできます。

[Row Filter]

```
public.large_table = id IN (SELECT id2 FROM temp_table)
```

行フィルターが正しく記述されていない場合、LiveCompare はフィルターを適用しようとして失敗します。したがって、この特定のテーブルの比較はスキップされ、例外がログファイルに書き込まれます。

テーブルが Row Filter セクションにリストされているが、何らかの理由で Table Filter によって除外された場合、このテーブルの行フィルターは黙って無視されます。

重要 : conflicts モードは行フィルターを利用しないことに注意してください。

行フィルターで現在のタイムスタンプを使用する

Row Filter は、`data_fetch_mode` によって異なる方法で適用されます。

- Postgres では、`data_fetch_mode` を `server_side_cursors_with_hold` または `server_side_cursors_without_hold` に設定すると、クエリが実行されたときにテーブル比較の先頭에만適用されます。これは、サーバー側のカーソルを使用してデータを取得することを意味します。
- Postgres では、`data_fetch_mode` を `prepared_statements` (デフォルト) に設定すると、準備されたクエリに Row Filter が含まれ、フェッチされるすべてのデータバッファーで実行されます。つまり、クエリが Row Filter で `now()`、`CURRENT_TIMESTAMP`、または `SYSDATE` (EPAS で) を使用している場合、準備されたステートメントが実行されると、Postgres は現在のタイムスタンプを再評価します。

したがって、Row Filter で `now()`、`CURRENT_TIMESTAMP` または `SYSDATE` を使用する場合：

```
[Row Filter]
public.table3 = logdate < CURRENT_TIMESTAMP
```

次に、サーバー側のカーソルを使用して、現在のタイムスタンプがクエリの先頭でのみ評価されるようにする必要があります。つまり、`data_fetch_mode` は `prepared_statements` とは異なる値に設定する必要があります。

Oracle では、上で説明したように、`data_fetch_mode` 設定は無視され、クエリが最初に実行され、クライアント側のカーソルを介してデータがフェッチされます。これにより、データは比較の開始時のスナップショットとして表示されます。これはクライアント側のカーソルですが、動作は Postgres でサーバー側のカーソルを使用する場合と似ています。

2.8.7 カラムフィルター

このセクションでは、列レベルのフィルターを任意のテーブルに適用できるため、LiveCompare は列フィルターの一部ではない列でのみ機能します。

このセクションには、1 行に 1 つのテーブルのリストを書くことができます (`schema_qualified_table_names` が無効になっていない限り、すべてのテーブル名はスキーマ修飾する必要があります)。たとえば、`public.table1` と `public.table2` の両方に列 `column1`、`column2`、`column3`、`column4` および `column5` があるとします。

```
[Column Filter]
public.table1 = column1, column3
public.table2 = column1, column5
```

この場合、テーブル `public.table1` の場合、LiveCompare は `column2`、`column4`、および `column5` 列でのみ機能し、`column1` および `column3` を除外します。一方、テーブル `public.table2` の場合、列 `column2`、`edb_tran column1` と `column5` を出します。

一般的な設定 `schema_qualified_table_names` を無効にしている場合は、接続の `start_query` 設定で Postgres に適切な `search_path` も設定する必要があります。

```
[General Setting]
...
schema_qualified_table_names = off

[My Connection]
...
start_query = SET search_path TO public

[Column Filter]
table1 = column1, column3
table2 = column1, column5
```

列フィルターに存在しない列名が指定されている場合、つまり、指定されたテーブルに列が存在しない場合、LiveCompare は有効な列がある場合のみを使用して、見つからなかった列に関するメッセージをログに記録します。

テーブルが **Column Filter** セクションにリストされているが、何らかの理由で **Table Filter** によって除外された場合、このテーブルの列フィルターは黙って無視されます。

重要： **Column Filter** で指定された列がテーブル PK の一部である場合、比較で無視されないことに注意してください。LiveCompare はそれをログに記録し、そのような列のフィルターを無視します。

重要： **conflicts** モードは列フィルターを利用しないことに注意してください。

2.8.8 比較キー

Column Filter と同様に、このセクションでは、テーブルごとに列のリストを指定することもできます。これらの列は、テーブルに主キーまたは **UNIQUE** 制約がある場合でも、特定のテーブルの比較キーと見なされます。

例：

```
[Comparison Key]
public.table1 = col_a, col_b
public.table2 = c1, c2
```

上記の例では、テーブル `public.table1` の比較キーは列 `col_a` および `col_b` になります。テーブル `public.table2` の場合、列 `c1` および `c2` は比較キーと見なされます。

上記の **Column Filter** セクションで説明されている、欠落している列、フィルターアウト、または欠落しているテーブルに関する同じ動作は、**Comparison Key**. 同様に、**Comparison Key** セクションは競合モードでは無視されます。

2.8.9 競合フィルター

このセクションでは、BDR ノードから競合を取得するときに `--conflicts` モードで使用するフィルタを指定できます。任意の SQL 条件式を作成し、式で以下のフィールドを使用できます。

- `origin_node` : サブスクリプションの上流ノード
- `target_node` : サブスクリプションのダウンストリームノード
- `local_time` : ノードで競合が発生したときのタイムスタンプ
- `conflict_type` : 競合の種類
- `conflict_resolution` : 適用された解像度
- `nspname` : 関係のスキーマ名
- `relname` : 関係するリレーションのリレーション名

セクションの下で `conflicts` 属性を使用する必要があります。以下の例を見つけてください。

```
[Conflicts Filter]
conflicts = conflict_type = update_missing AND nspname = my_schema
```

上記の構成を INI ファイルに追加することにより、LiveCompare は、各 BDR ノードで競合を照会中に、タイプが `update_missing` であり、スキーマ `my_schema` の下のテーブルに関連する競合のみを取得します。

重要 : このセクションは `--conflicts` モード専用であることに注意してください。

2.9 Licenses

2.9.1 TQDM

`tqdm` は共同作業の産物です。特に記載のない限り、すべての著者（コミットログを参照）はそれぞれの作品の著作権を保持し、MIT ライセンス（以下のテキスト）の下で作品をリリースします。

例外または著名な著者は、新しい順にリストされています。

- ファイル : *** MPLv2.0 2015-2020 (c) Casper da Costa-Luis**
`casperdcl`。
- ファイル : `tqdm/_tqdm.py` MIT 2016 (c) [PR #96] Google Inc.
- ファイル : `tqdm/_tqdm.py setup.py README.rst MANIFEST.in .gitignore` MIT 2013 (c) Noam Yorav-Raphael、元の著者。

[PR #96] : <https://github.com/tqdm/tqdm/pull/96>

Mozilla Public License (MPL) v. 2.0 - 別紙 A

このソースコードフォームは、Mozilla Public License, v. 2.0 の条件に従います。MPL のコピーがこのファイルと共に配布されなかった場合、<https://mozilla.org/MPL/2.0/> で入手できます。

MIT ライセンス (MIT)

Copyright (c) 2013 noamraph

このソフトウェアおよび関連するドキュメントファイル（「ソフトウェア」）のコピーを取得する人には、使用、コピー、変更、マージする権利を含むがこれらに限定されないソフトウェアを無料で許可します、ソフトウェアのコピーを発行、配布、サブライセンス、および/または販売し、次の条件に従って、ソフトウェアが提供される人にこれらを許可します。

上記の著作権表示およびこの許可表示は、ソフトウェアのすべてのコピーまたは実質的な部分に含まれるものとします。

本ソフトウェアは、「現状有姿」で提供され、明示または黙示を問わず、商品性、特定の目的への適合性、および非侵害の保証を含むがこれらに限定されません。作者または著作権所有者は、契約のいずれか、不法行為、またはその他の方法で、ソフトウェアまたは使用またはその他の取引に起因するかどうかを問わず、一切の責任を負わないものとしますソフトウェア。

2.9.2 cx_Oracle

CX_ORACLE のライセンス契約

Copyright © 2016, 2020, Oracle および/またはその関連会社。すべての権利を保有。

著作権 © 2007-2015、アンソニー・トゥイニング。すべての権利を保有。

Copyright © 2001-2007, Computronix (Canada) Ltd., エドモントン, アルバータ, カナダ. すべての権利を保有。

次の条件が満たされる場合、変更の有無にかかわらず、ソースおよびバイナリ形式での再配布と使用が許可されます。

ソースコードの再配布は、上記の著作権表示、この条件のリスト、および以下の免責事項を保持する必要があります。バイナリ形式での再配布は、上記の著作権表示、この条件のリスト、および次の免責事項を、配布時に提供されるドキュメントおよび/またはその他の資料に複製する必要があります。著作権所有者の名前も貢献者の名前も、特定の書面による許可がない限り、このソフトウェアから派生した製品を推奨または宣伝することはできません。免責事項：このソフトウェアは、著作権所有者および寄稿者によって「現状のまま」提供されるものであり、商品性および特定の目的への適合性の黙示的な保証を含みますが、これらに限定されません。いかなる場合においても、お客様またはコントリビューターは、直接的、間接的、偶発的、特別、例示的、または結果的な損害（代替品またはサービスの調達、データ、または損失、事業の中断）ただし、契約、厳格な責任、または不法行為（過失を含む）を問わず、いかなる方法でも、このソフトウェアの使用から発生した場合、およびその理論に基づいています。

Computronix[®] は、Computronix (Canada) Ltd の登録商標です。

© Copyright 2016, 2020, Oracle および/またはその関連会社。すべての権利を保有。一部著作権 © 2007-2015, Anthony Tuininga。すべての権利を保有。 Copyright © 2001-2007, Computronix (Canada) Ltd., エドモントン, アルバータ, カナダ. 著作権所有リビジョン 10e5c258。

Apache ライセンス

Apache License
Version 2.0, January 2004
<http://www.apache.org/licenses/>

使用、複製、および配布に関する条件

1. 定義。

「ライセンス」とは、本書のセクション 1 から 9 で定義される使用、複製、および配布の条件を意味します。

「ライセンサー」とは、ライセンスを付与する著作権所有者によって承認された著作権所有者またはエンティティを意味します。

「法的団体」とは、行動する団体と、その団体を管理する、管理される、または共通の管理下にある他のすべての団体の組合を意味するものとします。この定義の目的において、「支配」とは、(i) 契約であるかどうかにかかわらず、そのような事業体の指示または管理を引き起こす権限、または (ii) 50 % またはより多くの発行済み株式、または (iii) そのような事業体の実質的な所有権。

「あなた」（または「あなたの」）とは、このライセンスによって付与された許可を行使する個人または法人を意味します。

「ソース」フォームとは、ソフトウェアソースコード、ドキュメントソース、および構成ファイルを含むがこれらに限定されない、変更を行うための好ましいフォームを意味します。

「オブジェクト」フォームとは、ソースフォームの機械的な変換または変換の結果として生じるフォームを意味します。これには、コンパイルされたオブジェクトコード、生成されたドキュメント、および他のメディアタイプへの変換が含まれますが、これらに限定されません。

「作品」とは、ソースまたはオブジェクトの形式を問わず、作品に含まれるまたは添付される著作権表示で示される、ライセンスの下で利用可能な著作物を意味します（以下の付録に例を示します）。

「派生作品」とは、ソースまたはオブジェクトの形式を問わず、作品に基づいた（または派生した）作品であり、編集上の改訂、注釈、精緻化、またはその他の変更が全体としてオリジナルの作品を表すものとします。著作権の。このライセンスの目的において、派生作品には、作品およびその派生作品から分離可能な作品、または単にリンクする（または名前をバインドする）作品は含まれません。

「コントリビューション」とは、著作権所有者または個人または著作権所有者に代わって提出する権限を与えられた法人。この定義において、「提出された」とは、ライセンサーまたはその代表者に送信されるあらゆる形式の電子的、口頭、または書面によるコミュニケーションを意味します。作品の議論と改善を目的としてライセンサーによって、またはライセンサーに代わって管理されますが、著作権所有者によって「非寄稿」として明確にマークされているか、別の方法で指定されているコミュニケーションを除きます。

「コントリビューター」とは、ライセンサー、およびライセンサーがコントリビューションを受け取り、その後作品に組み込まれた個人または法人を意味します。

2. 著作権ライセンスの付与。このライセンスの条件を条件として、各コントリビューターは、あなたに、複製、派生作品、公に展示、公演、ソースまたはオブジェクト形式で作品およびそのような派生作品をサブライセンスし、配布すること。
 3. 特許ライセンスの付与。このライセンスの条件を条件として、各コントリビューターは、永久、世界的、非独占的、無料、ロイヤリティフリー、取消不能（このセクションに記載されている場合を除く）の特許ライセンスを付与します。作品の使用、販売、販売、輸入、およびその他の方法の移転そのような貢献が提出された作品。作品または作品に組み込まれた貢献物が直接的または貢献的な特許侵害を構成すると主張する団体に対して特許訴訟（訴訟での相互請求または反訴を含む）を提起した場合、そのためにこのライセンスに基づいてあなたに付与された特許ライセンス作業は、そのような訴訟が提起された日に終了するものとします。
 4. 再配布。次の条件を満たす場合、変更の有無にかかわらず、ソースまたはオブジェクトの形式で、作品またはその派生作品のコピーを複製および配布することができます。
 - (a) あなたは、作品または派生作品の他の受信者にこのライセンスのコピーを与えなければなりません。そして
 - (b) 変更されたファイルには、ファイルを変更したことを示す通知を表示する必要があります。そして
 - (c) あなたは、配布する派生作品のソース形式で、作品のソース形式からのすべての著作権、特許、商標、および帰属の通知を保持しなければなりません。作品; そして
 - (d) 作品に配布の一部として「NOTICE」テキストファイルが含まれる場合、配布する派生作品には、そのような NOTICE ファイルに含まれる帰属通知の読み取り可能なコピーを含める必要があります。派生物の一部、次の少なくとも 1 つの場所：派生物の一部として配布される NOTICE テキストファイル内。派生物とともに提供される場合、ソースフォームまたはドキュメント内。または、そのような第三者通知が通常表示される場合、派生物によって生成された表示内。NOTICE ファイルの内容は情報提供のみを目的としており、ライセンスを変更しないでください。あなたは、あなたが配布する派生物内に、作品からの NOTICE テキストと一緒に、またはその補遺として、あなた自身の帰属通知を追加することができます。ただし、そのような追加の帰属通知は、ライセンスを変更すると解釈されません。
- あなたは、あなたの変更あなた自身の著作権ステートメントを追加することができ、あなたの変更の使用、複製、または配布、またはそのような派生作品全体について、追加または異なるライセンス条件を提供できます。その他の点で、作品はこのライセンスに記載されている条件に準拠しています。
5. 寄付の提出。あなたが明示的に述べない限り、あなたが作品に含めるためにライセンサーに意図的に提出された貢献物は、追加の条件なしでこのライセンスの条件に従うものとします。上記にかかわらず、本契約のいかなる内容も、そのような投稿に関してお客様がライセンサーと締結した可能性のある別個の使用許諾契約の条件に取って代わったり、変更したりするものではありません。
 6. 商標。このライセンスは、作品の出所を説明し、NOTICE ファイルの内容を複製する際に合理的かつ慣習的に使用する場合を除き、ライセンサーの商号、商標、サービスマーク、または製品名を使用する許可を付与しません。
 7. 保証の否認。適用法で義務付けられている場合、または書面で同意しない限り、ライセンサーは作品を「現状のまま」提供します。タイトル、非侵害、商品性、または特定の目的への適合性の保証または条件。あなたは、作品の使用または再配布の妥当性を判断する責任を単独で負い、このライセンスに基づく許可の行使に関連するリスクを引き受けます。
 8. 責任の制限。いかなる場合も、いかなる法理論の下でも、不法行為（過失を含む）、契約、またはその

他の方法であるかどうかを問わず、適用される法律（故意および重大な過失行為など）で要求されるか、書面で合意された場合を除き、コントリビューターはあなたに対して責任を負いません。損害、このライセンスの結果、または作品を使用または使用できないことから発生した性質の直接的、間接的、特別、偶発的、または結果的な損害（のれん、作業の停止に対する損害を含むがこれらに限定されません）、コンピューターの障害または誤動作、またはその他すべての商業的損害または損失）、そのような投稿者がそのような損害の可能性について知らされていた場合でも。

9. 保証または追加責任の受け入れ。作品またはその派生作品を再配布しているときに、サポート、保証、補償、または本ライセンスに一致するその他の責任義務および/または権利を提供し、料金を請求することができます。ただし、そのような義務を受け入れる際に、他のコントリビューターを代表してではなく、自分自身に代わって唯一の責任において行動することができます。または、そのような保証または追加の責任を受け入れるという理由で、そのような寄稿者に対して主張された請求。

利用規約の終わり

2.9.3 Psycopg2

psycopg2 はフリーソフトウェアです。フリーソフトウェアファウンデーションが発行する GNU 劣等一般公衆利用許諾書、ライセンスのバージョン 3 または（あなたのオプションで）のいずれかの条件の下で、再配布および/または変更できます。

psycopg2 は役立つことを願って配布されていますが、保証はありません。商品性または特定の目的への適合性の黙示の保証でさえありません。詳細については、GNU Lesser General Public License を参照してください。

また、特別な例外として、著作権者は、本プログラムと OpenSSL ライブラリ（または OpenSSL と同じライセンスを使用する OpenSSL の改変バージョン）とをリンクさせ、両者を含むリンク先の組み合わせを配布することを許可しています。

OpenSSL 以外で使用されるすべてのコードについて、あらゆる点で GNU Lesser General Public License に従う必要があります。この例外を使用してファイルを変更する場合、この例外をファイルのバージョンに拡張できますが、義務はありません。これを望まない場合は、ご使用のバージョンからこの例外ステートメントを削除してください。プログラム内のすべてのソースファイルからこの例外ステートメントを削除する場合は、ここでも削除してください。

Psycopg2 とともに GNU Lesser General Public License のコピーを受け取っているはずです（doc/ディレクトリを参照）。そうでない場合は、<https://www.gnu.org/licenses/> を参照してください。

代替ライセンス

次の BSD に似たライセンスは、パターン `psycopg/adapter*.{h,c}` および `psycopg/microprotocol*.{h,c}` に従うファイルに（オプションで）適用されます。

商用アプリケーションを含むあらゆる目的でこのソフトウェアを使用し、次の制限に従って自由に変更および再配布することが許可されます。

1. このソフトウェアの出所を偽ってはなりません。オリジナルのソフトウェアを作成したと主張してはなりません。このソフトウェアを製品で使用する場合、製品ドキュメントでの謝辞があれば幸いです、必須ではありません。

2. 改変されたソースバージョンは、その旨を明示する必要があり、オリジナルのソフトウェアであると偽ってはなりません。
3. この通知は、ソース配布から削除または変更することはできません。

2.9.4 表にする

Copyright (c) 2011-2020 Sergey Astanin と寄稿者

このソフトウェアおよび関連するドキュメントファイル（「ソフトウェア」）のコピーを取得する人には、使用、コピー、変更、マージする権利を含むがこれらに限定されないソフトウェアを無料で許可します、ソフトウェアのコピーを発行、配布、サブライセンス、および/または販売し、次の条件に従って、ソフトウェアが提供される人にこれらを許可します。

上記の著作権表示およびこの許可表示は、ソフトウェアのすべてのコピーまたは実質的な部分に含まれるものとします。

本ソフトウェアは、「現状有姿」で提供され、明示または黙示を問わず、商品性、特定の目的への適合性、および非侵害の保証を含むがこれらに限定されません。作者または著作権所有者は、契約のいずれか、不法行為、またはその他の方法で、ソフトウェアまたは使用またはその他の取引に起因するかどうかを問わず、一切の責任を負わないものとしますソフトウェア。© 2020 GitHub, Inc.

2.9.5 OmniDB

MIT ライセンス

部分 Copyright (c) 2015-2019、OmniDB チーム

このソフトウェアおよび関連するドキュメントファイル（「ソフトウェア」）のコピーを取得する人には、使用、コピー、変更、マージする権利を含むがこれらに限定されないソフトウェアを無料で許可します、ソフトウェアのコピーを発行、配布、サブライセンス、および/または販売し、次の条件に従って、ソフトウェアが提供される人にこれらを許可します。

上記の著作権表示およびこの許可表示は、ソフトウェアのすべてのコピーまたは実質的な部分に含まれるものとします。

本ソフトウェアは、「現状有姿」で提供され、明示または黙示を問わず、商品性、特定の目的への適合性、および非侵害の保証を含むがこれらに限定されません。作者または著作権所有者は、契約のいずれか、不法行為、またはその他の方法で、ソフトウェアまたは使用またはその他の取引に起因するかどうかを問わず、一切の責任を負わないものとしますソフトウェア。