



EDB

LiveCompare

リリース 2.2.0

EDB

2022年06月16日

目次

第 1 章	はじめに	3
1.1	比較パフォーマンス	4
1.2	ユーザーのセキュリティに関する考慮事項	4
1.2.1	Release Notes	5
1.2.2	Requirements	30
1.2.3	Supported Technologies	32
1.2.4	Command-line Usage	33
1.2.5	Advanced Usage	38
1.2.6	BDR Support	42
1.2.7	Oracle Support	51
1.2.8	Settings	57
1.2.9	Licenses	71

© Copyright EnterpriseDB UK Limited 2019-2021 - 全著作権所有。

第1章 はじめに

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チェックできます。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。

1.1 比較パフォーマンス

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。

1.2 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。

1.2.1 Release Notes

2.2.0 (2022-06-14)

新機能

- `--dry-run` 実行モードを追加しました。これにより、ユーザーは比較セッションを実行せずに `.ini` 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 (LIV-142、RT78462)：
- 比較セッション中に受信したアボートメッセージは時系列で印刷されます
- 接続がタイブレーカーまたは信頼できるソースであるかどうか、および到達可能かどうかを含む接続のリストを表示します。
- **Table Filter** が構成されている場合は表示します。
- 比較に含まれるテーブルをリストします。これは、**Table Filter** 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、該当する場合は **Column Filter**、**Row Filter** および **Comparison Key** を示します。

改善

- x86_64 での SLES 12 のサポート (LIV-112)。
- x86_64 での SLES 15 のサポート (LIV-113)。
- Postgres `kwlist` のソースコード (LIV-141、RT80745) に従って予約語のリストを更新しました。
- メインの進行状況バーのパーセンテージが、整数ではなく浮動小数点数を使用するようになりました。以前は、数値が切り上げられていました (LIV-138、RT80745)。
- EPAS で `edb_redwood_date` が有効になっている場合、`date` 列は `timestamp` 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 `edb_redwood_date` 設定をチェックすることにより、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。
- ハッシュ不一致ログメッセージを **WARNING** から **DEBUG** に降格しました (LIV-145)。
- ログには、メッセージに `application_name` が含まれます (LIV-141)。
- `data_fetch_mode = prepared_statements` (デフォルト) の場合、Postgres または EPAS の **Row Filter** で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、`data_fetch_mode = server_side_cursors_with_hold` または `server_side_cursors_without_hold` も設定する必要があります (LIV-155)。
- 検出された発散の数が増加した場合に適用 DML スクリプトを生成するときのパフォーマンスが向上しました (LIV-159)。

バグ修正

- 配列列が比較キーとみなされると、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)。
- 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。比較にかかる時間が `min_time_between_heart_beats` よりも短いテーブルの場合、常にゼロを示していました (LIV-149、RT75805)
- 未処理の例外が原因で比較ワーカーがハングするコーナーケースを修正しました (LIV-140、RT80745)。
- 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)。
- `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 (LIV-150) 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。
- 複数のフィルタセクションで同じテーブルを設定できるようにします (LIV-156)。

2.1.0 (2022-03-31)

新機能

- Postgres- BDR 4 (LIV-131) のサポート。
- 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。
- 新しい設定 `comparison_cost_limit` と `comparison_cost_delay` は、0 より大きい場合、各ワーカーに `comparison_cost_limit` 行数を処理した後に `comparison_cost_delay` 秒 (たとえば、0.5) 昼寝をするように指示します。

その他の変更

- 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、`parallel_chunk_rows` のデフォルト値は 0 に設定されています。詳細については、設定 (LIV-130) のドキュメントを参照してください。
- `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。

バグ修正

- `timestamp(6)` データ型の Oracle と Postgres の比較で `ORA-01830` で失敗する問題を修正しました (LIV-127)。

2.0.0 (2022-02-15)

重大な変更

- パッケージの名前が `2ndq-livecompare` から `edb-livecompare` に変更されました。
- 実行可能ファイルは `2ndq-livecompare` から `livecompare` に名前が変更されました。

新機能

- **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、主キーまたはユニークインデックスのないテーブルで役立ちます (LIV-5)。
- 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)。
- 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` 可能な列を無視して、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、`ignore_nullable = false` (LIV-96) を設定することで無効にできます。
- 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、LiveCompare は一意性を強制するかどうかをチェックするようになりました。列セットを使用しているとき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、`check_uniqueness_enforcement = false` を設定することで無効にできます (LIV-119)。

パフォーマンスの向上

- Oracle 12c 以降では、LiveCompare は既に共通のハッシュを使用して `comparison_algorithm = block_hash` および `comparison_algorithm = row_hash` を使用できます。これらはより高速で、マルチプルのワーカプロセス間でテーブルを分割できます。これは、Oracle 12c で導入された `standard_hash()` 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 `sys.dbms_crypto` Oracle システムパッケージ (LIV-79) に対する `EXECUTE` 権限を持っている場合、同等の機能 `sys.dbms_crypto.hash()` を使用して、Oracle 11g で `comparison_algorithm = block_hash` および `comparison_algorithm = row_hash` も許可します。

その他の改善

- Oracle 21c (LIV-47) のサポートを追加しました。
- LiveCompare スキーマをレプリケーション対応 (BDR、pglogical またはネイティブ論理レプリケーション) データベースに追加できるようになりました (LIV-42)。
- LiveCompare は、PostgreSQL 14 の新しい `pg_read_all_data` ロールを使用できます (LIV-73)。
- 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。

バグ修正

- DML 生成時のエラーを回避するために、ユニコードシーケンス `\u0000` を適切に引用します。
- テーブルを複数のラウンドに分割した場合、合計行数が正しく表示されない問題を修正しました (LIV-14)。
- Oracle の空の BLOB と Postgres の空の `bytea` を比較すると誤検出が発生する問題を修正しました (LIV-103)。
- 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。LiveCompare は再接続して、可能な場合は比較を続行できるようになりました (LIV-84)。
- 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)。
- テーブルが接続に存在せず、`logical_replication_mode` が無効になっている場合、テーブル比較が中止されない問題を修正しました (LIV-108)。
- `timestamp` データ型のフィールドが常に Oracle と Postgres の間で不一致のハッシュを生成していた問題を修正しました。
- 無視された列が一般的なハッシュでまだ考慮されている問題を修正しました。

非推奨の通知

- Oracle 10g (LIV-47) へのサポートを終了しました。

1.18.1 (2021-12-14)

バグ修正

- 非同期接続で内部の丸い部分の列名設定が考慮されていない問題を修正しました (LIV-95)。

1.18.0 (2021-12-01)

パフォーマンスの向上

- Oracle 用の新しい QueryBlock データフェッチメソッドを実装しました。アルゴリズムは同じソケットから行をフェッチし続けます。Oracle では、以前は最後に処理されたマルチ列の PK 値を過ぎて行をフェッチする必要があった行値式を含む不等号演算子が許可されないため、より効率的です。これにより、Oracle と Postgres の比較のパフォーマンスが大幅に向上します。PostgreSQL では行値式を含む不等式演算子が許可されているため、LiveCompare は準備されたステートメントを使用して同じアルゴリズムを使用して、膨張と xmin ホライズンの抑制の観点から影響を与えません。したがって、Postgres と Postgres のパフォーマンスはこの実装の影響を受けません (LIV-83、RT75644、RT75675、RT75332)。
- `comparison_algorithm = block_hash` を使用するときバッファをリセットする必要性を削除しました。ハッシュが一致しないブロックが見つかった場合、LiveCompare は、不一致のブロック (既にフェッチされている) 内のすべての行が個別に比較されるまで正しく `comparison_algorithm = row_hash` に切り替え、次のバッファをフェッチするために `comparison_algorithm = block_hash` に戻ります。これにより、Oracle と Postgres、および Postgres と Postgres の比較 (LIV-85、LIV-86、RT75644、RT75675、RT75332) のパフォーマンスが向上します。
- `comparison_algorithm` が `block_hash` または `row_hash` (現在 Oracle 12 以降が必要) に設定されている場合、LiveCompare は Oracle と Postgres の比較でも大きなテーブルを分割できるようになりました。ハッシュはデータベース側で行われるため、以前は Postgres 対 Postgres でのみ可能でした。Oracle と Postgres は共通のハッシュ関数を共有していないため、MD5 に基づく共通の行ハッシュを整数ハッシュに変換して、テーブルデータをマルチプルのワーカーに分割できるようにしました。Oracle では、パフォーマンス上の理由からデータベース側で実行できなかったため、このハッシュを Python 側で計算します。Oracle と Postgres のテーブル分割の並列処理のコストは、Postgres と Postgres の場合と同じではないため、Oracle と Postgres では、`parallel_chunk_rows` をデフォルトの 1000 万よりも高く設定することをお勧めします。たとえば、1 億以上に設定すると、アルゴリズムは非常に大きなテーブルのみを分割します (LIV-87)。

その他の改善

- EDB Extended PostgreSQL 14 および EDB Advanced PostgreSQL 14 (LIV-71、LIV-72) のサポート。
- 相違の検査に役立つ Oracle rowid を収集する (LIV-90、RT75644、RT75675)。
- 相違の検査に役立つ LiveRound 比較アルゴリズムのログを改善しました (LIV-91、RT75644、RT75675)。

バグ修正

- `oracle_fetch_fk_metadata` が尊重されない問題を修正しました (LIV-92)。
- ログが多いユースケースで `log_level = debug` を使用すると、ログキューがいっぱいだったために LiveCompare が未処理の例外で終了する可能性がある問題を修正しました。Unix システムでは、キューの最大サイズは 32676 です。基本の Python 実装がログレコードを非同期にエンキューしたために問題が発生しました。解決策は、Python 実装をオーバーライドしてログレコードを同期的にエンキューすることでした。これは、`log_level = debug` およびログが多いユースケース (LIV-93)

1.17.0 (2021-10-01)

の改善

- PostgreSQL 14 (LIV-66) のサポートを追加しました。
- Debian 11 (LIV-70) のサポートを追加しました。

バグ修正

- `--recheck` は常にセッション ID を必要とします (LIV-76)。
- ドキュメントのマイナーな修正 (LIV-26、LIV-68、LIV-80)。

1.16.0 (2021-08-04)

パフォーマンスの向上

- Oracle と Postgres の比較セッションで、一般的なユースケースのパフォーマンスを向上させる MD5 に基づいた共通の行ハッシュを使用できるようになりました。以前は、Oracle と Postgres の比較は `comparison_algorithm = full_row` に制限されていましたが、現在は `comparison_algorithm = row_hash` または `comparison_algorithm = block_hash` も使用できます。デフォルトでは、`comparison_algorithm = block_hash` に設定されています。Oracle の制限により、共通ハッシュ、BLOB、および CLOB フィールドの使用は 2000 文字に制限され、行全体の幅は 4000 文字を超えることはできません。共通ハッシュの要件を満たさないテーブルは、`comparison_algorithm = full_row` を使用して個別に比較できます。詳細については、ドキュメントを確認してください (RT72038、LIV-61)。
- `buffer_size` のデフォルト値を 256 行から 4096 行に増やしました。
- `parallel_chunk_rows` のデフォルト値を 100 万行から 1000 万行に増やしました。

1.15.0 (2021-06-06)

の改善

- `--conflicts` 実行モードの競合フィルタリングを導入しました (LIV-40)
- LiveCompare は、照合が指定されていない場合、混合テクノロジーの比較 (Oracle と PostgreSQL など) で `force_collate = C` を想定します (LIV-44)。
- 既に追加された EPAS 13 へのサポートに加えて、EPAS 11 および 12 (LIV-48) へのサポートを追加しました。

バグ修正

- Oracle と PostgreSQL の比較で LiveCompare が動かなくなった (RT71243 / LIV-43)
- `--conflicts` で作成された比較セッションで `--recheck` を実行できないバグを修正しました (LIV-50)

1.14.0 (2021-05-14)

新機能

- BDR 3 クラスターに固有の新しい `--conflicts` 実行モード。BDR 3.6 の場合、テーブル `bdr.apply_log` がチェックされ、BDR 3.7 の場合、テーブル `bdr.conflict_history` がチェックされます。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、既に PK 値でフィルタリングされている影響を受けるテーブルでのみ自動クラスター全体の比較を実行できます。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。BDR の競合のみをチェックすることにより、比較は `--compare` モードよりもはるかに高速であるため、頻繁に実行してクラスターの一貫性を確保することが実行可能であり、推奨されます。

の改善

- LiveCompare は、特定のテーブルが存在しない場合、またはテーブルがレプリケーションセットの構成に従ってレプリケートされたデータを受信しない場合に、BDR ノードを無視できるようになりました。以前は比較はすべての接続で実行されましたが、比較コンセンサスは必要に応じてテーブルごとにノードを自動的に除外できます。コンセンサスマジョリティは、テーブルが存在し、テーブルがノードがサブスクライブしているレプリケーションセットに関連付けられているノード間で引き続き達成されます。これにより、BDR Witness ノード (LIV-29) を適切に無視することを含め、レプリケーションセットによって定義された異なるアーキテクチャを持つ BDR クラスターで比較できます。
- すべての実行モードの短い説明を表示する `--help` のメッセージを改善しました (LIV-28)。
- 列が NULL 可能かどうかに関する情報もテーブルのメタデータに収集されるようになりました (LIV-28)。

- ドキュメントで、列が主キーの一部である場合、ソートできないデータ型が **string** に変換されることを明確にしました (LIV-4)。
- **oracle_ignore_unsortable** が Oracle の PK の一部ではないソートできない列を無視することをドキュメントで明確化しました (LIV-26)。
- **data_fetch_mode** が **prepared_statements** (デフォルト) または **server_side_cursors_with_hold** (LIV-26) に設定されている場合、**parallel_data_fetch** を有効にできる (デフォルトで) ことをドキュメントで明確にしました。
- **max_parallel_workers** のデフォルト値が2であることをドキュメントで明確化しました (LIV-26)。

バグ修正

- LiveCompare は型をスキーマ修飾名として扱うようになりました。この修正の前に、LiveCompare は **search_path** 外のタイプを見つけることができませんでした (LIV-35)。

非推奨

- BDR 3.7 で非推奨になった **bdr.difference_fix*** 関数の使用を削除しました。これらの関数は PostgreSQL スーパーユーザではなく **bdr_superuser** を必要とするため、BDR 3.6 用に生成された DML スクリプトにまだ含まれています。これはドキュメント (LIV-1、LIV-31) で明確にされています。

1.13.1 (2021-04-14)

バグ修正

- Oracle データ型 **RAW** および **ROWID** のサポートの問題を修正しました (RT69656)。
- Oracle からのバイナリオブジェクトフィールドが **NULL** の場合の未処理の例外を修正しました (RT69656)。
- ドル引用符で囲まれた文字列を含む Oracle x Postgres 比較サポートの問題を修正しました (RT69656)。
- 特定の PostgreSQL の予約キーワードとして名前付けられた列が引用され、比較で考慮されないコーナーケースを修正しました (RT69656)。

1.13.0 (2021-02-25)

パフォーマンスの向上

- PostgreSQL 接続にのみ適用される新しい設定 **data_fetch_mode** を実装しました。次のいずれかの値を想定できます。
 - **prepared_statements** : このバージョンおよび以前のバージョンのデフォルトの動作。準備されたステートメントを使用してデータを取得します。データベース上のフットプリントが小さく、一般的なユースケースで優れたパフォーマンス。運用データベースには **strongly** をお勧めします。
 - **server_side_cursors_with_hold** : サーバー側のカーソル **WITH HOLD** を使用して

データを取得します。これにより、データの各チャンクが LiveCompare に送信される前に一時ファイルで具体化されるため、xmin が抑制され、データベースのメモリと一時ファイルの使用量が増加します。その結果、複数の小さなテーブルの方が高速になる可能性があり、テストと移行のシナリオに推奨されます。- `server_side_cursors_without_hold` : サーバー側のカーソル WITHOUT HOLD を使用してデータをフェッチします。xmin を保持できますが、データベースのメモリと一時ファイルの使用量は WITH HOLD よりも低くなります。大規模なテーブルでは高速になる可能性があるため、負荷の低いデータベースにお勧めします。

その他の改善

- 新しい設定 `abort_on_setup_error` を実装しました。有効にすると、テーブルの比較ラウンドを設定するときにエラーが発生した場合に比較セッション全体を中止するように LiveCompare に指示します。デフォルトで無効になっています。つまり、LiveCompare は何らかの理由でセットアップフェーズが失敗したテーブルをログに記録し、無視します。
- 新しい設定 `abort_on_setup_error` のため、設定 `setup_max_attempts` および `setup_min_interval_between_attempts` は廃止されました。

1.12.0 (2021-02-11)

新機能

- 新しい設定 `session_replication_role` を実装しました。有効にすると、適用スクリプトの先頭で `session_replication_role` を `replica` に設定します (RM20247、RT69742)。
- 新しい設定 `split_updates` を実装しました。有効にすると、UPDATE ステートメントを対応する DELETE および INSERT ステートメントに分割します (RM20355、RT69742)。
- 新しい接続設定 `fetch_fk_metadata`、特定の接続で外部キーに関するメタデータを取得するかどうかを指示します。デフォルトで有効になっています。
- Oracle 接続の `fetch_fk_metadata` の値をオーバーライドする、新しい一般設定 `oracle_fetch_fk_metadata`。デフォルトで無効 (RM20244、RT69656)。
- 新しい一般設定 `float_point_round`。浮動小数点値を比較するときに指定された 10 進数で丸めるように LiveCompare に指示します。デフォルトで無効になっています (RT69381)。

削除された機能

- `--fix` 実行モードを削除しました。ユーザーは、LiveCompare が生成する DML スクリプトを確認することをお勧めします。

の改善

- LiveCompare で生成された適用スクリプトには、先頭に `difference_fix_start_query` も含まれ、DML を適用する前にテーブルの所有者を設定するようになりました。
- `comparison_algorithm = full_row` は `parallel_chunk_rows` を尊重し、すべての接続で比較されるテクノロジーが PostgreSQL の場合、テーブルを複数のワーカーに分割します。テーブル分割を無効にするには、`parallel_chunk_rows = 0` を設定します。
- EPAS を比較するときにシステムスキーマ `sys` のテーブルを除外する (RM20392)。
- `max_parallel_workers` のデフォルト値を 2 から 4 に増やしました。

バグ修正

- `comparison_algorithm = block_hash` で、パラレルワーカーが処理していたデータチャンクに相違がある場合のパフォーマンスの低下を修正しました。
- `custom_dollar_quoting_delimiter` のデフォルトのランダムに生成された文字列が原因で、FIPS 対応オペレーティングシステムで LiveCompare がクラッシュする問題を修正しました (RT70032、RM20378)。
- `oracle_ignore_unsortable` または `column_intersection` がすべての列を削除しても、LiveCompare が比較を実行しようとする問題を修正しました。現在、LiveCompare はこれらのテーブルでの比較を中止し、適切なメッセージをログに記録します。
- LiveCompare が `custom_dollar_quoting_delimiter` を使用して PK 値を表していなかったコーナーケースを修正しました (RT69656)。
- `--recheck` モードで LiveCompare を実行すると、順序付けが異なるために出力スキーマの制約が損なわれる可能性があるバグを修正しました。

1.11.0 (2021-01-19)

の改善

- LiveCompare によって生成される DML コマンドの順序の新しい設定 `difference_statement_order`、デフォルトで `delete_update_insert` に設定されます (RT69742、RM20245)。
- EnterpriseDB PostgreSQL Advanced Server のサポート - EPAS 13。
- CentOS および RHEL システムでの PostgreSQL 13 のパッケージ化。

バグ修正

- 最小 PK 値が既にデータの最後に達したカーソルの最大 PK 値より大きいためにカーソルがまだ初期化されていない場合、以前のカーソルは初期化されますが、そのデータは終了 (RT69742、RM20249)。
- LiveCompare が `logical_replication_mode = pglogical` のときに `Initial Connection` と `node_name` を許可していないバグを修正しました (RM19795)。
- LiveCompare が `schema_qualified_table_names = off` のときに強制するのではなく、`schema_qualified_table_names = on` のときに強制的に `oracle_user_tables_only = on` を強制していたバグを修正しました (RT69656)。
- `comparison_algorithm = full_row` の場合、文字列が適切にクォートされないバグを修正しました (RM20251)。
- 生成された DML で文字列が適切に引用されないバグを修正しました。

1.10.1 (2020-10-29)

バグ修正

- 初期メタデータフェッチフェーズで行われる、各テーブルの最小および最大プライマリキー値のフェッチのパフォーマンスを向上しました (RM19681、RT69088)。
- 比較セッションを再開でき、実行モード `--fix` および `--recheck` は、同じメジャーバージョンの場合、別の LiveCompare バージョンで以前に作成した比較セッションに対して実行できます。例：セッションが 1.10.0 で作成され、顧客が LiveCompare を 1.10.1 にアップグレードしてから `--fix` を実行した場合、セッションは同じメジャーバージョンで作成されたため、LiveCompare は `--fix` モードの実行を許可します (RM19686、RT69088)。
- LiveCompare メタデータで `Row Filter` を保存するときの一重引用符の問題を修正しました (RM19682、RT69088)。
- LiveCompare のデフォルトのドルクォート区切り文字 (`custom_dollar_quoting_delimiter` が設定されていない場合) が修正されたハッシュ文字列になりました。実行間でランダム設定値が変更されたため、比較セッションを再開できませんでした (RM19684、RT69091)。
- `custom_dollar_quoting_delimiter` の設定を `off` で明示的に無効にできるようになりました。つまり、LiveCompare はデフォルトのハッシュ文字列を区切り文字として使用します。 (RM19685、RT69091)。

1.10.0 (2020-10-02)

の改善

- 新しい一般設定 `custom_dollar_quoting_delimiter`、Postgres DML コマンドを生成するときに文字列を引用する方法を LiveCompare に指示します。設定されていない場合 (デフォルト)、LiveCompare はランダムな文字列を生成してドル引用符の区切り文字として使用します (RT68950、RM19159)。
- LiveCompare は、セッションの最初の実行 (セッション作成) と次の実行 (`resume`、`--recheck` または `--fix`) の間に設定も接続属性も変更されていないことをチェックするようになりました。
- バックグラウンドまたは別のシェルウィンドウで実行中に LiveCompare をソフトに停止できるようになりました (RM18564)。

バグ修正

- ユーザーが `apply_on_the_<node>.sql` ファイルを使用して手動で修正を適用した後、`--recheck` モードで実行中に LiveCompare が差分の数を正しく更新していませんでした。
- LiveCompare を `--recheck` モードで複数回実行すると、`apply_on_the_<node>.sql` ファイルに複製されたエントリが表示され、サマリーに違い/修正の複製数が表示されます。
- サマリービューで多数の相違が発生すると、`integer out of range` エラーが発生する。

1.9.1 (2020-08-13)

バグ修正

- 特定のプラットフォームに存在しない可能性のある POSIX 定数を処理しました (RM18197)。
- パーティション分割されたテーブルを扱う場合、ユーザーがテーブルフィルターを設定してパーティションを削除すると、LiveCompare はマスターテーブルとフィルター処理されたものを除く他のすべての子の両方を比較します。これは修正されました。 (RM16912)。

1.9.0 (2020-08-06)

新機能

- 設定 `full_comparison_mode = on|off` は非推奨になり、新しい `block_hash` アルゴリズム設定を許可する新しい設定 `comparison_algorithm` に置き換えられました-これにより、基本的なテストで 50 % のパフォーマンスが向上しました。サポートされているアルゴリズムの完全なリストは次のとおりです。 - `full_row` (古い `full_comparison_mode = on` と同じ) : ハッシュを使用した行比較を無効にします。また、ハッシュに依存するため、テーブル分割も無効になります。設定 `parallel_chunk_rows` は無視され、テーブルは分割されません。この場合、完全な比較は、行を列ごとに比較することによって実行されます。PostgreSQL 以外のデータ接続がある場合、LiveCompare は自動的に `comparison_algorithm = full_row` を設定します。 - `row_hash` (旧 `full_comparison_mode = off` と同じ) : ハッシュを

使用した行比較を有効にし、テーブル分割を有効にします。テーブルは分割されるため、各ワーカーはテーブルごとに最大 `parallel_chunk_rows` を比較します。データ行は PostgreSQL でハッシュされるため、比較は `full_row` よりも高速です。ただし、特定の行についてハッシュが一致しない場合、その特定の行について、LiveCompare は `full_row` アルゴリズムにフォールバックします（つまり、行ごとに比較します）。この設定は、すべてのデータ接続が PostgreSQL の場合にのみ許可されます。- `block_hash`（新しく実装された比較アルゴリズム）: `row_hash` と同じように動作しますが、行ごとに比較する代わりに、LiveCompare は「ブロックハッシュ」、つまり、フェッチされたデータバッファ内のすべての行のハッシュ（最大 `buffer_size` 行）。概念的には、2 レベルのマークルツリーのように機能します。ブロックハッシュが一致する場合、LiveCompare はブロック全体を進めます（これが、この比較アルゴリズムが `row_hash` よりも高速です）。ブロックハッシュが一致しない場合、LiveCompare は `row_hash` にフォールバックし、バッファ内で行ごとに比較を実行して、発散する行を見つけます。この設定は、すべてのデータ接続が PostgreSQL の場合にのみ許可されます。これがデフォルト値（RM14146）です。

- LiveCompare は、エラーで終了する前に、各テーブルセットアップで数回試行できるようになりました。この動作を構成するために、2 つの新しい設定が追加されました。 `setup_max_attempts`（デフォルトは 3）と `setup_min_interval_between_attempts`（デフォルトは 30 秒）（RM17518）。

改善

- LiveCompare の出力は、> を使用して出力をリダイレクトしているときに出力ファイルに印刷され、ssh を介して LiveCompare を実行しているときにも表示されるようになりました。
- ログとデバッグを改善するためにコンセンサスがリファクタリングされました（RM17107）。
- 比較セッションの最後に、LiveCompare は、ログレベルに応じて、検出された問題のカウンターを表示するようになりました（RM18196）。
- Tie の場合、すべてのノードに対して DML が生成されるようになりました（RM16656）。
- LiveCompare は、特定のテーブルで相違が見つかった場合、テーブルの概要に無視される列の最大数を表示するようになりました（RM16581）。
- LiveCompare は、Output または Initial（指定されている場合）の適切なメッセージで中止されるようになりました。接続に到達できません。また、到達可能なデータ接続が 2 つ未満の場合は、適切なメッセージで中止します。2 つ以上到達可能な場合、その 1 つだけを比較します（RM14030）。
- Debian および Ubuntu パッケージには `python3-setuptools` が必要になりました（RM17232）。
- サポートされている最小の Python バージョンと Linux ディストリビューションに関するドキュメントを明確化しました（RM17232）。
- パーティション分割されたテーブルを扱う場合、LiveCompare はマスターテーブルを使用する代わりに各パーティションをスキャンすることを優先します。このようにして、行数のより良い見積もりと、ワーカー間のジョブのより良い分割を実現します。
- 差分カウントの代わりに POSIX 標準のリターンコードを使用する（RM18197）。
- TPAexec を使用して LiveCompare をインストールする方法を示すドキュメントに、サンプル `config.yml` の例を追加しました（RM15312）。
- `--recheck` および `--fix` モードのロギングを改善しました（RM15768）。

- 例外が発生した場合、ログに Python スタックトレースが含まれます (RM17158)。
- PgBouncer を介して PostgreSQL に接続する LiveCompare に関するいくつかの考慮事項をドキュメントに含めました (RM18019)。
- `vw_running_processes` ビューに `round_id` および `round_part` フィールドを追加しました。これは、LiveCompare の実行が cron ジョブとしてスケジュールされている場合の比較の進行状況を確認するのに役立ちます。

バグ修正

- `Output Connection` で `tuple concurrently updated` を引き起こした Consensus の競合状態を修正しました (RM17107)。
- `num_processed_rows` が実際の値よりも高く報告される問題を修正しました (RM17373)。
- 実行中に何らかの問題が見つかった場合、LiveCompare は `successfully executed` を報告しなくなりました (RM17289)。
- ユーザーが `max_parallel_workers <= 0` を設定した場合にハングしないように、少なくとも 1 人のワーカーを強制しました。
- 印刷されたテーブルの大きな整数の固定番号形式。LiveCompare は、科学表記法を使用する代わりに数値全体を表示するようになりました (RM16683)。
- `--fix` モード実行後に `--recheck` モードが誤った差分ステータスを報告する問題を修正しました (RM15953)。
- 無視された列のリストがテーブルのメタデータで誤って報告される問題を修正しました (RM17663)。
- 別の信頼できるソースまたはタイブレーカーで LiveCompare を再実行すると、相違が同点として誤って報告される問題を修正しました (RM17909)。

1.8.0 (2020-06-15)

新機能

- 新しい設定 `difference_sources_of_truth`、常にコンセンサスを得る必要がある接続を指示するために使用されます。 `consensus_mode` を `source_of_truth` に設定する必要があります (RM15952)。
- セッションフォルダーが作成される場所を示す新しい設定 `work_directory`。cron ジョブとしてスケジュールされた LiveCompare を実行するのに役立ちます (RM16476)。

改善

- 差分修正 DML スクリプトに書き込まれる DML コマンドの順序は、外部キーを考慮に入れるようになりました。--fix 実行モードでも同じ順序が使用されます (RM15766)。
- LiveCompare は、既知の接続 ID のリストに対して difference_tie_breakers および difference_sources_of_truth をチェックするようになりました。不明な接続 ID が指定された場合、ユーザーにエラーが表示され、比較が中止されます (RM16622)。
- CentOS 8 および Ubuntu 20.04 のサポートを追加しました (RM14480、RM14619、RM14974)。
- 通常のテーブルを印刷するときに、データが null の場合に空のセルが印刷される。代わりに [null] を出力するようになりました (RM16686)。

バグ修正

- Coverity スキャナーで検出された軽微な問題を修正 (RM16623)。
- メインスクリプトが3つのパラメーターのみで呼び出されたときに正しく処理し、セッション ID か実行モードかを正しく区別します。
- 表データを転置形式で印刷するときに、データが null の場合、フィールド全体が省略される。この問題は修正されました (RM16686)。

1.7.0 (2020-05-18)

重大な変更

- 実装された LiveCompare 実行モード：---compare：デフォルトの実行モード。比較のみを実行し、並列で実行されているスレッドを再確認/修正することに違いはありません。比較中に、検出された各違いは、後で再チェックとオプションの自動修正のために difference_log テーブルに保存されます。・ --recheck：compare モードで作成済みのセッションに対して何度でも実行できます。一つ一つ違いを再確認し、difference_log テーブルを更新します。・ --fix：compare モードで作成済みのセッションに対して実行できます。違いを1つずつ再確認し、自動的に修正を試み、difference_log テーブルを更新します。
- 比較モードでは、相違のリストを保持するビューは vw_open_differences ですが、再確認および修正モードの場合、ビューは vw_differences です。
- 自動修正モードで、BDR >= 3.6.18 の場合、LiveCompare は bdr_local_only_origin と呼ばれる新しい作成された複製起点を使用します。
- 設定 difference_mode、min_time_between_difference_checks、max_difference_check_attempts および difference_check_nap_time を削除しました。
- 設定 live_mode、min_time_between_rounds、および max_tail_rounds_before_full_round を削除しました。
- show_progressBars の設定が有効になっている場合 (デフォルトです)、Python モジュール tqdm が < 4.16.0、LiveCompare はアップグレード方法を説明して中断します。

新機能

- デフォルトで無効になっている新しい一般設定 `column_intersection` を実装しました。この設定を有効にすると、PK 列が同じである限り、LiveCompare は異なる列セットを含むテーブルの比較を許可します。比較で考慮される列のセットは、すべての接続のテーブルに存在する列の共通部分です (RM14147、RT67064)。
- 新しいセクション **Column Filter** を実装しました。ここでは、テーブルごとに、比較で無視する列のコンマ区切りのリストを定義できます。PK の一部である列は無視できません。このセクションのフォーマットは、**Row Filter** セクション (RM14629、RT67064) と同様に、1 行に 1 つのテーブルです。
- デフォルトで無効になっている新しい一般設定 `oracle_ignore_unsortable` を実装しました。有効にすると、テーブルに PK がない場合、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 (RT67064)。
- デフォルトで無効になっている新しい一般設定 `oracle_user_tables_only` を実装しました。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。また、**Table Filter** -> `tables` はスキーマ名なしでテーブル名でフィルタリングできるようになりました (RT67064)。
- デフォルトで有効な新しい一般設定 `schema_qualified_table_names` を実装しました。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。 (RM14901、RT67042)。
- `schema_qualified_table_names` が有効で `start_query` が設定されていない場合 (デフォルト)、LiveCompare は CVE-2018-1058 (RM15391) から保護するために `start_query` を使用して `search_path` をクリアします。
- 新しい一般設定 `force_collate` を実装しました。デフォルトでは `off` に設定されます。これは、PostgreSQL で照合が強制されないことを意味します。有効な照合名に設定すると、照合が異なる Postgres データベース、または Postgres に C 以外の照合がある場合は Oracle と Postgres データベースを比較すると役立ちます (この場合、ユーザーは `force_collate = C` を設定する必要があります) (RM15016、RT67064)。
- デフォルトで無効になっている新しい一般設定 `fetch_row_origin` を実装しました。この設定を有効にすると、LiveCompare は各行の BDR/pglogical オリジン名を取得します (RM14487)。
- 比較中に例外が見つかった場合、LiveCompare は特定のテーブルの比較ラウンドを中止し、新しい列 `rounds.round_error` にエラーを書き込み、失敗したワーカーをプールに戻します。
- LiveCompare の一般的な進行状況バーに、比較中にエラーが発生したために中止されたテーブルの数が表示されるようになりました。

改善

- 最初に接続を開き、データベース接続を再利用します (RT67042)。
- ユーザーに設定ファイルを読み取る権限がない場合のエラーメッセージを明確にしました (RT67064)。
- Table Filter と Row Filter には一般設定 `schema_qualified_table_names` が無効になっていない限りスキーマ修飾テーブル名が必要であることをドキュメントで明確化し、構成例を提供しました (RM15311、RT67669)。
- 構成ファイルに認識されない設定が見つかった場合、エラーメッセージで中断します。
- 必要な接続数についてドキュメントで明確化。
- CentOS / RHEL の EPEL リポジトリへの依存関係に関するドキュメントを明確化。
- postgres オペレーティングシステムユーザー用の `cx_Oracle` Python モジュールのインストール方法に関するドキュメントを明確化。
- セッションディレクトリとログファイルに `lc_` プレフィックスを追加しました。
- いずれかの接続で `technology = oracle` と Python モジュール `cx_Oracle` が見つからない場合、LiveCompare は現在のユーザーの最新の `cx_Oracle` をインストールする方法の説明を中止します。
- 自動生成された DML *.sql スクリプトには、BDR の `SET LOCAL bdr.xact_replication = off;` 句が含まれています。

バグ修正

- 列名が Postgres の予約語であるため、列名の不一致が誤って評価され、Oracle と Postgres のテーブル比較が中止される問題を修正しました (RT67064)。
- Oracle: テキスト列を含む PK が `ORA-00920: invalid relational operator` エラーを生成することがある問題を修正 (RT67064)
- ソートできないデータ型の列が `ORDER BY` 句で適切に処理されない Oracle と Postgres の問題を修正しました (RT67064)。
- 設定 `difference_allow_null_updates` が誤って解釈されていたため、`UPDATE` が列を `NULL` に設定していない DML ジェネレーターの問題を修正しました。
- 比較カーソルが適切に進行しないコーナーケースを修正しました。
- ユーザーが既に終了した比較セッションを再開しようとしたときのエラーメッセージを修正しました。

1.6.0 (2020-04-03)

- PostgreSQL ≥ 11 の場合、`hashtext` の代わりに `hashtextextended` を使用して比較ワーカー間でデータを分割します (RT67167、RM13664)。
- Oracle PK の列名のリストに重複する列名があり、エラーが発生して比較が中止される問題を修正しました (RT67064、RTM14145)。
- 最初にデータ接続に接続できない場合、LiveCompare は比較セッション全体を中止するようになりました (RT67042)。
- ラウンド状態が頻繁に保存され、`min_time_between_round_saves` を尊重しない問題を修正しました。パフォーマンスを向上させます。
- 接続とクエリの問題のログを改善しました。

1.5.0 (2020-03-13)

- Oracle は `parallel_data_fetch` をサポートしていないため、接続の 1 つが Oracle の場合、`parallel_data_fetch` の設定は自動的に無効になります。
- 実行できるラウンドの最大数と処理できる行と 1 回の比較で見つかる差分を増やしました (データ型は `integer ~ bigint`) (RM13664、RT67167)。
- プログレスバーを ASCII から Unicode に変更。
- 5 秒ごとに経過したグローバルプログレスバーを更新します。
- グローバル進行状況バーからレートと推定時間を削除し、接続数を追加しました。
- Python モジュール TQDM ≥ 4.16 が利用可能な場合、グローバル進行状況バーに追加の統計 (検出された違いと適用された自動修正) が表示されます。
- テーブルが分割されていても、最大プロセス数が比較予定のテーブル数によって制限される問題を修正した。
- テーブルのメタデータを取得するときのログの冗長性を INFO から DEBUG に減らしました。
- 手動 (Ctrl-C) または自動割り込み (`stop_after_time`) が発生したログケースを含みます。

1.4.0 (2020-02-13)

- 多方向比較のパフォーマンスを向上させるために、並列データフェッチを実装しました。パラレルデータフェッチはデフォルトで有効になっていますが、`parallel_data_fetch = off` を設定して無効にできます。
- 行がローカル、つまり `xmin` がレプリケーションオリジンに関連付けられていない問題を修正しました (RT66906)。

1.3.0 (2020-01-30)

- 多方向比較: LiveCompare は任意の数の接続を比較できるようになりました。比較は、すべてのノードから同時にデータを取得することによって実行されます。これにより、コンセンサス（クォーラムベースと単純過半数の両方がサポートされます）とタイブレーカー接続のオプションのリストに基づいて、データの不整合を判断できます。新しいビュー `vw_consensus` を使用して、LiveCompare がどのように機能したかの詳細を確認できます。
- 新しい一般設定 `all_bdr_nodes` を追加しました。有効にすると、ユーザーは BDR ノードをポイントする `Initial Connection` セクションのみを指定でき、LiveCompare は BDR クラスター内のすべてのアクティブノードを考慮して接続のリストを構築します。`bdr.node_summary` ビューで表示されるように、LiveCompare がノード DSN を使用してすべての BDR 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。
- データ比較で相違が見つかったときにどの接続（または BDR ノード）が正しいかを決定する新しい一般設定 `consensus_mode` を追加しました。`simple_majority` または `quorum_based` です。`consensus_mode = quorum_based` の場合、新しい設定 `difference_required_quorum` (0.0~1.0 の値、デフォルト 0.5) が考慮されます。デフォルトは `consensus_mode = simple_majority` です。
- 新しい設定 `difference_tie_breakers` を追加しました。データに相違がある場合にコンセンサスが正しい接続またはノードを決定できない場合に役立ちます。接続名のコンマ区切りリストである必要があります。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` ファイルまたは `bdr_all_nodes = on` で定義する必要があります、`Initial Connection` のみが定義されていますが、`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。
- 多元比較では、接続部名に「Left」「Right」以外の接続名も可能です。下位互換性が保たれているため、ユーザーは引き続き `Left Connection` と `Right Connection` を定義できますが、上記で説明したように `difference_tie_breakers` の定義が必要になるのは 2 つの接続のみです。以前は、「左」接続は常にタイブレーカーと見なされました。つまり、自動差分修正が有効になっている場合は正しいと見なされました。
- 多方向比較では、接続の 1 つだけが PostgreSQL と異なる必要があります。
- 新しい設定 `stop_after_time` を実装して、数秒後に LiveCompare を手動で中断できます。デフォルトでは `stop_after_time = 0` これは、LiveCompare が自動的に停止しないことを意味します（手動 Ctrl-c のみ）。LiveCompare は、すべての場合に Ctrl-c を使用して手動で停止できます。LiveCompare が手動で中断されたか自動で中断されたかに関係なく、コマンドラインでセッション ID を引数として渡すことで再開できます。
- 新しいテーブル `difference_fix` は、各データ接続で実行された（または実行しようとした）正確な DML LiveCompare、時間、エラー（ある場合）を保存します。スクリプト `applied_*.sql` にも、同じように適用された DML が含まれています。
- LiveCompare はテーブルの所有者をテーブルのメタデータに保存するようになりました。
- 差分修正トランザクションのタイムスタンプを 2000-01-01 から 2010-01-01 に変更しました。これは BDR < 3.6.11、3.6.11 以降、組み込み関数 `bdr.difference_fix_xact_set_avoid_conflict` が使用されるため。
- 自動修正が無効になっている場合、BDR データベースに `bdr_livecompare` レプリケーションオリジンが不必要に作成されていた問題を修正しました。

- 比較ワーカプロセスが正常に終了しない場合がある問題を修正しました。
- CI : LiveCompare パッケージが Ubuntu 18.04 LTS 用にもビルドされるようになりました。

1.2.0 (2020-01-03)

- LiveCompare は、Oracle と Postgres の比較で BLOB フィールドをサポートするようになりました。
- 相違点の無視：ユーザーは、出力 PostgreSQL 接続 (RM8939) 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、特定の相違点の相違の再チェックを停止できます。
- 揮発性のダイバージェンス：違いの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 は再チェックを停止し、違いを揮発性としてマークします (RM10964)。
- 上書きされた相違：自動修正が適用された後、再チェック時に `xmin` が変更された場合、修正後に行が変更されたことを意味します。LiveCompare は、発散を上書き済みとしてマークします (RM10964)。
- 修正不能な相違：自動修正が正常に適用された後、再チェックしても `xmin` が変更されていない場合、LiveCompare は相違を修正不能としてマークします (RM10964)。
- LiveCompare は、相違がない場合に `code = 0` を返し、相違がある場合にリターンコード `> 0` を返すようになりました。
- テーブルフィルターと行フィルターは、すべての一般設定とともにテーブル `<livecompare_schema_name>.settings` に保存されるようになりました。
- 新しい一般設定 `difference_check_nap_time`、新しい差分チェックスプリントを開始する前に差分チェックワーカーがスリープする秒数を制御します。デフォルト：5 秒。
- BDR 3 のテーブルリストを構築するとき、LiveCompare は宣言型パーティションを考慮しません (RT66502)。
- LiveCompare は、PostgreSQL 接続に対してのみ DML スクリプトを生成するようになりました。
- 自動修正を有効にする場合 (`difference_mode = live_fix` または `difference_mode = offline_fix`)、右の接続が PostgreSQL である必要があります。
- 修正されたセキュリティ問題：テーブル/列の名前とすべてのリテラルが適切に引用されます (RM12530)。
- カラム名に予約語を使用した場合の不具合を修正しました (RM12530)。
- ビュー `vw_differences` の列名の問題を修正しました (RM12529)。
- ハッシュ比較でテーブルが不必要に分割されていた問題を修正しました。

1.1.0 (2019-12-18)

- BDR 3 のテーブルリストを構築するときに、LiveCompare は、Left Connection と Right Connection の両方の BDR ノードに関連付けられたレプリケーションセットの共通部分のみを考慮するようになりました。
- パーティション分割されたテーブルまたはパーティションが検討されているときにテーブルリストを構築する際の問題を修正しました (RT66499)。
- 接続の検証とテーブルリストの作成の初期ステップのログの詳細度を向上させました。
- Oracle のエンコーディングと文字列の処理に関する問題を修正しました。

1.0.0 (2019-12-03)

- LiveCompare は BDR でレプリケーションオリジンを作成して使用できます。BDR バージョンが 3.6.11 以降の場合、LiveCompare は、`bdr_superuser` 権限を持つユーザーまたは PostgreSQL スーパーユーザーが BDR ファンクション `bdr.difference_fix_origin_create(text)`、`bdr.difference_fix_session_setup(text)`、`bdr.difference_fix_session_reset()`、および `bdr.difference_fix_xact_set_avoid_conflict()` を使用してレプリケーションオリジン管理を実行する必要があります。BDR バージョンが 3.6.10 以下の場合、LiveCompare は PostgreSQL の機能を利用した複製起点管理を行うために PostgreSQL のスーパーユーザーが必要です。それ以外の場合、LiveCompare はレプリケーションオリジンを管理しようとしません。(RT66192 および RM11000)。
- LiveCompare は、BDR/pglogical の一貫性のない各行から複製元情報を取得できます。BDR バージョンが 3.6.11 以降の場合、LiveCompare は各行からレプリケーション起点情報を取得するために `bdr_superuser` 権限を持つユーザーまたは PostgreSQL スーパーユーザーを必要とします。BDR バージョンが 3.6.10 以下または pglogical 3 を使用している場合、LiveCompare には各行からレプリケーション元の情報を取得するための `pglogical_superuser` 権限を持つユーザーまたは PostgreSQL スーパーユーザーが必要です。それ以外の場合、LiveCompare は複製起点情報を取得しようとしません (RM11971)。
- 親テーブルが既にテーブルリストにある場合、テーブルリストからパーティションを削除します (RT65920 および RM10994)。
- pglogical と BDR のテーブルリストを構築するときは、常にレプリケーションセットテーブルを検討します (RT65920)。
- 空の文字列を処理するときの問題を修正しました (RT65918 および RT65988)。
- 0.11.0 から 0.12.0 へのアップグレードの問題を修正しました (RT65918 および RT65988)。
- 最小 PK と最大 PK 値の決定の問題を修正しました。

0.12.0 (2019-11-01)

- 自動修正トランザクションで、ロールをテーブルの所有者に変更するように `difference_fix_start_query` 設定のデフォルト値を変更しました。これは、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、LiveCompare が使用する特権ロールにアクセスするのを防ぐために行われます。その結果、右の接続のユーザーは、ロールをテーブルの所有者に切り替える必要があります (RM11000)。
- LiveCompare がテーブルリストを構築した後、LiveCompare がテーブルの比較ラウンドを開始する前に、テーブルが削除されるか、重大なスキーマの変更を受け取った場合に処理されました。LiveCompare は、比較ラウンドの前にテーブルに関するメタデータをチェックおよび更新するようになりました (RT65918)。
- `pg_replication_origin_session_setup` 実行時の問題を修正しました (RT65988)。
- 最大 PK 値の決定の問題を修正しました (RM11340)。
- データベースに多数のテーブルがある場合に、テーブルからメタデータを取得するときのパフォーマンスの問題を修正しました。

0.11.0 (2019-10-10)

- 各差分チェックがテーブル `difference_log` に記録されるようになりました。これは、時間の経過に伴う各差分の分析に役立つ情報を提供します。各違いは、次のいずれかのステータスを通過できます。
 - **検出 (D)** : 違いが検出されました。 `difference_mode = live_nofix` または `difference_mode = live_fix` の場合、LiveCompare は、結論に至るまで (以下の他のステータスを参照)、最大 N 回 (`max_difference_check_attempts` の設定で構成可能)、各再チェックの間に X 秒間待機します (また `min_time_between_difference_checks` を設定して構成できます)。 `difference_mode = offline_nofix` または `difference_mode = offline_fix` の場合、見つかった各違いはすぐに **永続的** と見なされます。
 - **永続的 (P)** : `max_difference_check_attempts` 回の違いを再チェックした後、LiveCompare は再チェックを停止し、違いを **永続的** としてマークします。 If `difference_mode = offline_nofix` or `difference_mode = offline_fix`, then all differences are marked as permanent at the moment they are detected, because there is no re-check.
 - **Absent(A)** : If before having reached `max_difference_check_attempts`, LiveCompare finds that the difference does not exist anymore (the row is now consistent between both databases), then LiveCompare stops re-checking and marks the difference as **Absent** (in previous versions, LiveCompare would remove the register from the difference table).
 - **NotAllowed(N)** : The difference was detected, but LiveCompare is forbidden to automatically fix this difference because user has limited the types of differences that can be automatically fixed (via settings `difference_statements` and/or `difference_allow_null_updates`).
 - **Fixed(F)** : The difference was automatically fixed by LiveCompare in the Right Connection, by applying the DML from field ``right_difference_differtran19``
- 出力接続の自動スキーマ変更: ユーザーが `Output Connection` を以前の LiveCompare バージョンで使われたデータベースをポイントしている場合、LiveCompare は自動的にスキーマの変更を処理します。0.11.0 の場合、LiveCompare はスキーマを再作成しますが、0.12.0 以降、実装はスキーマの変更のみを適用し、ユーザーデータを保持します。
- `.ini` ファイルからのセッション値を保存するテーブル `connections` および `settings` を作成しました。

- 各違いの行表現の改善：行全体が JSON として保存されるようになりました。
- 異なる各行の拡張列に `xmin` を格納します。
- 各行の拡張列 (`ctid`、`xmin`、`origin`) も JSON として保存されるようになりました。
- 複数行の値を持つテキスト列、バイナリデータを持つ列、および配列列の DML を改善しました。
- テーブルに PK がなく、複数行の文字列、null 値、または型が `bytea` の場合の未処理の例外を修正しました。または、テーブルに PK があった場合、ただし、PK に上記の状況のいずれかがある場合。

0.10 (2019-09-24)

- 新しい設定 `difference_mode` を追加しました。違いは報告されますが、修正されません。
`offline_fix` : 比較中のテーブルは負荷がかかっていないため、差分は再チェックされません。違いが見つかったら、`Right Connection` で報告されて修正されます。
`-live_nofix` (デフォルト) : LiveCompare は比較中のテーブルに負荷がかかっていると想定するため、LiveCompare はそれらを再チェックして、結果一貫性のためになくなったかどうかを確認します。永久的な違いが報告されていますが、修正されていません。
`-live_fix` : LiveCompare は、比較中のテーブルに負荷がかかっていると想定するため、LiveCompare はそれらを再チェックして、結果一貫性のためになくなったかどうかを確認します。違いは、永続的としてマークされると報告され、修正されます。
- `difference_recheck` の設定が `difference_mode` にマージされました。 `difference_mode = offline_nofix` にマップされた `difference_recheck = off` および `difference_mode = live_fix` にマップされた `difference_recheck = on` の動作。
- 新しいグローバル設定 `difference_statements` を追加しました。これは、`difference_mode = offline_nofix` または `difference_mode = live_nofix` の場合に LiveCompare によって DML スクリプトに生成されるか、または `difference_mode = offline_fix` または `difference_mode = live_fix` の場合に自動的に適用されるかを制御します。 `difference_statements` の値は次のとおりです。
- `UPDATE SET col = NULL` のようなコマンドを差分レポートまたは自動修正で許可するかどうかを決定する、新しいグローバル設定 `difference_allow_null_updates` (デフォルトは `on`) を追加しました。
- 新しいグローバル設定 `difference_fix_replication_origin` を追加しました。手動で設定されていない場合、`pglogical 3` および/または `BDR 3` 比較のデフォルトで `bdr_livecompare` に設定されます。LiveCompare は、特定のレプリケーションオリジンが存在しない場合は作成し、`difference_mode = live_fix` または `difference_mode = offline_fix` の場合はこのレプリケーションオリジンを使用してすべての自動 DML 修正を適用します。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、`SELECT pg_replication_origin_drop('<value of setting difference_fix_replication_origin>');` を使用してレプリケーションオリジンを削除できます。
- 新しいグローバル設定 `difference_fix_start_query` を追加しました。これは、各トランザクションの先頭に実行されて、右側の接続の違いを自動的に修正します。BDR 3.6.7 以降では、`difference_fix_start_query` が空の場合、LiveCompare は自動的に `difference_fix_start_query = SET LOCAL bdr.xact_replication = off;` を設定します。また、LiveCompare は `difference_fix_start_query` を自動的に設定して、差分修正トラン

ザクションが `difference_fix_replication_origin` で指定された複製起点を使用するようにします。

- 接続が開かれるたびに任意のクエリを実行するために使用できる新しい接続設定 `start_query` を追加しました。
- 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する新しいグローバル設定 `show_progressBars` (デフォルトは `on`) を追加しました。バッチ実行に役立ちます。
- Postgres の比較で、見つかった各違いは行の `ctid` も保存するようになりました。BDR 3 または pglogical 3 が使用されている場合、検出された各違いには、行の `xmin` のレプリケーション起点も保存されます。
- 生成された DML スクリプトは常にすべての DML を単一のトランザクション内に配置します。`difference_fix_start_query` が (手動または自動で) 定義されている場合、トランザクションの先頭に追加されます。
- 実行の最後にグローバル進行状況バーが削除されない問題を修正しました。
- 出力データベースの存在がチェックされていないバグを修正しました。

0.9 (2019-08-30)

- 左または右の接続での Oracle データベースのサポート。Oracle Instant Client と Python モジュールは個別にインストールする必要がありますが、Postgres データベースには必要ありません。LiveCompare は、Oracle に接続していなくても機能します。
- 現在、行ハッシュとテーブル分割ハッシュは PostgreSQL と PostgreSQL の比較でのみ許可されています。いずれかの接続で PostgreSQL 以外のテクノロジーが使用されている場合、新しい設定 `full_comparison_mode` は自動的に `on` に設定されます。Postgres と Postgres でもハッシュの使用を無効にしたい場合は、構成ファイルで `full_comparison_mode = on` を明示的に定義できます。
- BDR 1 および 2 のサポート。 `logical_replication_mode = bdr` の場合、`node_name` で接続を定義し、`replication_sets` でテーブルをフィルタリングできます。
- Left Connection と Right Connection の両方の接続が PostgreSQL にある場合、行ハッシュは `md5()` である必要があります。11. それ以外の場合、両方の接続で `hashtextextended()` を使用します。これにより、混合 PostgreSQL バージョンの比較が可能になります (たとえば、9.4 と 12)。
- `logical_replication_mode` を設定すると、PostgreSQL バージョンと拡張機能の存在の検証が Initial Connection および Left Connection に適用されます。テーブルリストは Left Connection から構築されていることに注意してください。ただし、Right Connection では、接続のみがチェックされます。これにより、混合テクノロジーの比較 (PostgreSQL と Oracle、BDR と PostgreSQL、BDR 2 と BDR 3 など) が可能になります。
- ユーザーが差分再チェックを有効または無効にできる新しい設定 `difference_recheck` (ブール値、デフォルトは `on`)。
- テーブルスキーマの違い (列名と列のデータ型) は、後で分析するためにレポートデータベース (テーブル `tables`) に記録されます。

- 左右の接続情報は、後で分析するためにレポートデータベース（テーブル sessions）に記録されています。
- テーブルに重複する行がある場合の差分チェックのバグを修正しました。
- テーブルの右側の接続により多くの行がある場合の差分レポートのバグを修正しました。
- 一時的な差異がある場合の差異報告のバグを修正しました。

0.8 (2019-08-07)

- テーブルに複数のフィールドを持つ PK がある場合の単一行のフェッチを修正しました（一貫性をチェックするため）。

0.7 (2019-08-06)

- 行のないテーブルの処理を修正しました。
- 空のセクションの処理を改善しました。

0.6 (2019-06-17)

- ログコンポーネントを変更しました。

0.5 (2019-06-17)

- バグ修正： - 順序付け演算子なしのデータ型の並べ替えのサポート。 - hashtextextended() が利用できない場合に md5() をレコードハッシュとして使用する (PG <= 10)。 • 設定ファイルが存在しない場合、適切なメッセージを表示する。

0.4 (2019-06-12)

- 2ndQuadrant CI パイプラインに含めるための準備。

0.3 (2019-05-29)

- 接続を指定する DSN のサポート。
- 改善されたテーブルと行のフィルター。
- さまざまな種類の論理レプリケーションのサポート： - ネイティブ論理レプリケーション - pglogical - bdr

- BDR のサポート - ユーザーが接続のノード名を指定できるようにします。 - ユーザーが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できるようにします。
- さまざまなテストシナリオを作成しました。

0.2 (2019-05-21)

- ハッシュの改善: `md5()` の代わりに `hashtext()` と `hashtextextended()` を使用。
- フェッチはプリペアドステートメントを使用して実行されます。

0.1 (2019-05-17)

- PostgreSQL の初期サポート。
- スタンドアロンモードの初期実装。

1.2.2 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、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、次を参照してください。

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、次を参照してください。

<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
```

TPAexec との LiveCompare

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

- Name: livem1node2
```

(次のページに続く)

```
location: main
node: 2
role: replica
upstream: livemlnode1
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/

1.2.3 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、BDR ノードに関する情報を収集するために初期接続とともに使用されます。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	9.4	Data
PostgreSQL	9.5, 9.6, 10, 11, 12, 13 and 14	Data and/or Output
EDB PostgreSQL Extended	9.6, 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

次のページに続く

表 1 – 前のページからの続き

Technology	Versions	Connections
BDR	1, 2, 3 and 4	Initial, Data and/or Output
Oracle	11g, 12c, 18c, 19c and 21c	A single Data connection

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。

1.2.4 Command-line Usage

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。

- プロセスの数
- 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` になっている場合、テーブルが分析されていることを意味します（必要に応じて行数と分割を確認します）
- 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 |          |
+-----+-----+-----+-----+-----+

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:
```

(次のページに続く)

(前のページからの続き)

```
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;
```

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 が誤検知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したか、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に対してのみ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。

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。

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)

+-----+-----+-----+-----+-----+				
↪+-----+-----+-----+-----+-----+				
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	-	-	-	
+-----+-----+-----+-----+-----+				

1.2.5 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る DML コマンドを表示し、他のすべてのデータベースの一貫性を保ちます。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)

```
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** データベースの不一致を修正するには、 **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。

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 はキャッシュデータベースの `connections` テーブルにフラグ `connection_reachable` を書き込みます。

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て、`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、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。

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する **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）。

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。

差分ログ

テーブル `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。
- **不在 (A)** : 再チェック時に違いが存在しないことが LiveCompare によって検出された場合（行は両方のデータベースで一貫しています）、LiveCompare は違いを**不在**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
);
```

1.2.6 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 を使用してそれらに接

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)

```
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に注意してください。

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 Witnesses 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。

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、ローカルのための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ている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、次のこと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するため、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。

レプリケーション起点を含む上記のすべての手順は、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、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るため、トランザクションが過去に設定される (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;
```

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 クラスタに固有です。BDR 3.6、BDR 3.7、または BDR 4 クラスタでのみ機能します。

`compare` モードはテーブル全体のすべての内容を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、自動競合解決によってクラスタ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、影響を受けたテーブルでのみクラスター全体の比較を実行できます。既に 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
```

LiveCompare を conflicts モードで実行するには：

```
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 ）内に、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)

```
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 スクリプトを実行する場合:

```
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リングするように 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
```

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。

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。

1.2.7 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
```

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、そのままの状態ですべての 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 が Oracle Database に接続できるように、Python モジュール

cx_Oracle 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
```

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。

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 に指示することにより、この動作を回避できます。

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

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。

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;
```

1.2.8 Settings

一般設定

- `logical_replication_mode` : プログラムが接続とテーブルフィルタ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`
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 個の接続を開いたままにします。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 つすべてのモード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、このフェッチ方法で優れたパフォーマンスが得られますが、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サイドカーソル 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、こ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で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>
xmin hold	very low	medium	high
xmin released per	buffer	chunk	whole comparison session
temp files	very low	very high	low
memory	very low	high	low
allows async conns	yes	yes	no
fastest for	general	small tables	large tables
recommended load	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最小数。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_bitmaps = 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
```

(次のページに続く)

``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。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、各ワーカーが昼寝をせずに行を処理することを意味します。
- `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は、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ている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。有効にするには、logical_replication_mode を pglogical または bdr に設定する必要があります。
- ・ 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 を設定することにより、その動作をオーバーライドすることができます。これにより、LiveCompare は比較でヌル可能な列を考慮します。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場合、または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と (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 `session_replication_role` PostgreSQL 設定を出力適用スクリプトで `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 スーパーユーザーが必要です。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。

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`。

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`。

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 接続で使用されるサービス名。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。

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
```

(次のページに続く)

(前のページからの続き)

```
[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。

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）。

[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ませんが失敗します。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 `Row Filter` がテーブル比較の先頭のみ適用されます。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。

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` を除外しますが、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]
```

(次のページに続く)

```
table1 = column1, column3  
table2 = column1, column5
```

列フィルターで列名が指定されていない場合、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、有効な列のみを使用して無視します。

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。

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、列の欠落、除外された、またはテーブルの欠落に関する同じ動作が、**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。

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。

1.2.9 Licenses

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通知は、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。いかなる場合でも、作者または著作権所有者は、契

約の行為、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、責任を負わないものとしますソフトウェア。

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。

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

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の保証を含みますが、これらに限定されません。いかなる場合においても、Regents または Contributors 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、いかなる方法であれ、このソフトウェアの使用から生じたもの、および責任の理論に基づいて、通知された場合でも。

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。部分著作権 © 2007-2015、Anthony Tuininga。すべての権利を保有。Copyright © 2001-2007, Computronix (Canada) Ltd., Edmonton, Alberta, Canada. 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。

「法的機関」とは、行動する団体と、その団体を支配する、支配される、または共通の支配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、「制御」とは、(i) 契約であるか

どうかにかかわらず、そのようなエンティティ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」) 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 (例は以下の付録にあります)。

「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、作品に基づいて (または派生し)、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作品を意味します著作権の。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「非投稿」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、公演するための永久、世界的、非独占的、無料、ロイヤリティフリー、取消不能の著作権ライセンスを付与します。作品およびそのような派生作品をソースまたはオブジェクト形式でサブライセンスし、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に従い、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エンティティ (訴訟での相互請求または反訴を含む) に対して特許訴訟を提起した場合、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。あなたは、次の条件を満たすことを条件として、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、お客様が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。派生作品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、派生作品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作品内に、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、あなたの変更に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対して、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、本契約の内容は、そのような投稿に関してライセンサーと締結した別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、適用法（故意および重大な過失行為など）で要求されるか、書面で合意されない限り、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、間接、特別、偶発的、または結果的な損害（のれん、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単独の責任で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

Psycopg2

psycopg2 はフリーソフトウェアです。フリーソフトウェア財団が発行する 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 の改変版）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。そうしたくない場合は、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。

誰でもこのソフトウェアを、商用アプリケーションを含むあらゆる目的で使用し、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。

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通知は、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。いかなる場合でも、作者または著作権所有者は、契約の行為、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、責任を負わないものとしますソフトウェア。© 2020 GitHub, Inc.

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通知は、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。いかなる場合でも、作者または著作権所有者は、契約の行為、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、責任を負わないものとしますソフトウェア。