



EDB

LiveCompare

リリース 2.5.1

EnterpriseDB Corporation.

2024 年 05 月 23 日

目次

第 1 章	Comparison performance	3
1.1	ユーザーのセキュリティに関する考慮事項	3
第 2 章	Release notes	5
2.1	LiveCompare 2.5.1 release notes	5
2.2	LiveCompare 2.5 release notes	6
2.3	LiveCompare 2.4 release notes	7
2.4	LiveCompare 2.3 release notes	7
2.5	LiveCompare 2.2 release notes	7
2.6	LiveCompare 2.1 release notes	9
2.7	LiveCompare 2.0 release notes	10
第 3 章	Requirements	13
3.1	TPAexec を使用した LiveCompare	13
第 4 章	Supported technologies	15
4.1	PgBouncer のサポート	16
第 5 章	Command-line usage	17
5.1	比較モード	17
5.2	再チェックモード	19
5.3	競合モード	20
5.4	ドライランモード	20
第 6 章	Advanced usage	23
6.1	比較の中止	24
6.2	比較キー	25
6.3	修正する違い	25
6.4	差分ログ	26
第 7 章	EDB Postgres Distributed support	27
7.1	PGD 監視ノード	29
7.2	PGD クラスターの違い	30
7.3	PGD の競合	31
7.3.1	競合の例	32
7.3.2	競合フィルター	35
7.4	ミキシングテクノロジー	35
第 8 章	Oracle support	37
8.1	要件	40
8.1.1	Oracle インスタントクライアント	40
8.1.2	cx_Oracle Python モジュール	41

8.2	の違い	41
8.3	BLOB および CLOB データ型	41
8.4	互換性のない照合順序	42
8.5	共通ハッシュ	42
第 9 章	Settings	45
9.1	一般設定	45
9.2	初期接続	52
9.3	出力接続	52
9.4	データ接続	53
9.5	テーブルフィルタ	54
9.6	行フィルタ	56
9.6.1	行フィルタで現在のタイムスタンプを使用する	57
9.7	列フィルタ	57
9.8	比較キー	58
9.9	競合フィルター	59
第 10 章	Licenses	61
10.1	TQDM	61
10.1.1	Mozilla Public LicenseMPL v. 2.0 - 証拠 A	61
10.1.2	MIT ライセンス MIT	61
10.2	cx_Oracle	62
10.2.1	Apache ライセンス	62
10.3	Psycog2	65
10.3.1	代替ライセンス	65
10.4	集計する	66
10.5	OmniDB	66

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。このツールはデータベースを比較し、比較レポート、相違点のリスト、および便利な DML スクリプトを生成するため、オプションで DML を適用して、データベースの不一致を修正できます。

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、複数のテーブルマルチプルのワーカープロセスを同時にチェックでき、高度な構成が可能であるため、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ます。

各データベースの比較は、**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、プログラムは停止し、すべてのレポートを生成します。コンテキスト情報を失うことなく LiveCompare を開始および停止できるため、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 はテーブルを複数の比較ラウンドに分割し、これらも並列実行され、他のワーカーによって同時に実行されている他のテーブルと並行して実行されます。

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、テーブルの先頭最も古い既存の PK からテーブルの最後に新しい既存の PK まで開始します。ラウンドの開始後に挿入された新しい行は無視されます。LiveCompare は PK 列をソートして、各テーブルから最小および最大 PK を取得します。ソート不可の 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` を使用してこれを実現します。Oracle では、`to_char` を使用します。

比較アルゴリズムを実行する場合、各ワーカーには $N+1$ 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、出力/レポートデータベースへの接続であり、プログラムキャッシュも保持され、比較セッションを停止および再開できます。

比較アルゴリズムによって見つかった差異は、後の都合の良いときに手動で再チェックできます。レプリケーションの一貫性チェックを可能にするためにこれを行うことをお勧めします。差分が再チェックされると、レプリケーションがその特定の行に追いついた可能性があり、差分はもう存在しないため、差分は削除されます。それ以外の場合は、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 スクリプトを生成するため、それを確認して、違いを 1 つずつ修正できます。または、DML スクリプト全体を適用して、すべての永続的な差異を修正できます。

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。

- データベーステクノロジーの移行 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 x 新しいサーバー。
- 物理レプリケーションプライマリ x 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します。
- 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。
- 論理レプリケーション。Postgres ネイティブ論理レプリケーション、pglogical、および EDB Postgres DistributedPGD、以前は BDR として知られている 3 種類の論理レプリケーションテクノロジーがサ

ポートされています。

第1章 Comparison performance

LiveCompare は、実稼働システムで使用するように最適化されており、チューニングのためのさまざまなパラメーターを備えています。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、1秒あたり約77,000行、または4時間未満で10億行の比較パフォーマンスを提供しました。

この使用例は一般的な使用例です。低負荷、テスト、移行、およびその他の特定のシナリオでは、サーバーサイド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により、速度を向上できる場合があります。私たちの実験では、各種類のサーバーサイドカーソルは、小さいまたは大きいテーブルが関係するユースケースでパフォーマンスの向上を提供します。

1.1 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` ロールを持つユーザーが必要です。`logical_replication_mode = pglogical` の場合、LiveCompare には `pglogical_superuser` ロールを持つユーザーが必要です。

PGD で DML スクリプトを適用するには、すべての分岐接続潜在的にすべてのデータ接続で、`bdr_superuser` ロールを持つユーザーが `bdr.xact_replication` を無効にする必要があります。

PGD が使用されている場合、LiveCompare はすべての固定行を `bdr_local_only_origin` と呼ばれるレプリケーションオリジンに関連付けます。LiveCompare は、かなり過去のトランザクション日時を含む DML も適用するため、データベースで実行されている実際の DML と PGD 競合がある場合、LiveCompare DML は常に競合に負けます。

デフォルト設定の `difference_fix_start_query` では、適用スクリプトのトランザクションはテーブルの所有者にロールを変更して、データベースユーザーが悪意のあるトリガーを書いて修正を適用するロールへのアクセスを取得しないようにします。その結果、分岐接続のユーザーは、ロールをテーブル所有者に切り替えることができる必要があります。

第2章 Release notes

LiveCompare ドキュメントでは、マイナーリリースとパッチを含む LiveCompare 2 の最新バージョンについて説明しています。このセクションのリリースノートは、各リリースの新機能に関する情報を提供します。

Version	Release Date
LiveCompare 2.5.1 release notes	May 16 2024
LiveCompare 2.5.1 release notes	May 09 2023
LiveCompare 2.4 release notes	Nov 29 2022
LiveCompare 2.3 release notes	Aug 15 2022
LiveCompare 2.2 release notes	Jun 14 2022
LiveCompare 2.1 release notes	Mar 31 2022
Mozilla Public LicenseMPL v. 2.0 - 証拠 A	Feb 15 2022

2.1 LiveCompare 2.5.1 release notes

リリース日 2024 年 5 月 16 日

LiveCompare 2.5.1 には、次の新機能、機能強化、バグ修正、およびその他の変更が含まれています。

Type	Description	Support Ticket
Platform Support	LiveCompare は、x86 ベースの Debian 12 プラットフォームでの使用がサポートされるようになりました。	
Bug Fix	LiveCompare が null の代わりに bytea 列の設定として「None」を使用して DML を生成する問題を修正しました。	93247/25921

[次のページに続く](#)

表 2 – 前のページからの続き

Type	Description	Support Ticket
Bug Fix	Oracle 10g データベースに接続する機能を復元し、このような接続に `comparison_algorithm = full_row` を強制的に設定します。 EDB は、Oracle 10g での LiveCompare のテストを行っていないことに注意してください。そのため、LiveCompare で公式にサポートされている Oracle バージョンのリストには含まれません。Oracle 10g は以前にサポートされており、ほとんどの場合で引き続き動作すると予想されます。ただし、一部の制限が存在する場合があります。	99815/31762
Bug Fix	EPAS クエリと Oracle クエリによって返された主キーの順序の違いによって発生した問題を修正しました。	
Bug Fix	特定の列名が Oracle の予約語として誤って識別される問題を修正しました。	102498/34229

2.2 LiveCompare 2.5 release notes

リリース日 2023 年 5 月 9 日

LiveCompare 2.5 には、次の新機能、機能強化、バグ修正、およびその他の変更が含まれています。

Type	Description
Feature	EDB Postgres 分散 5 のサポート。
Enhancement	LiveCompare は、引用符で囲まれた識別子を適切に処理するために使用される、Oracle および Postgres の予約語のバージョンに依存しないリストを保持するようになりました。
Bug fix	HandleIgnoredColumns メソッドが混合比較モードで失敗していた問題を修正しました。
Bug fix	カスタム照合を使用するときに PrepareMask メソッドが失敗していた問題を修正しました。
Bug fix	`node_name` を使用すると、LiveCompare が PGD 4 および 5 の pg 論理メタデータを試行する問題を修正しました。

2.3 LiveCompare 2.4 release notes

リリース日 2022 年 11 月 29 日

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

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

2.4 LiveCompare 2.3 release notes

リリース日 2022 年 8 月 15 日

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

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

2.5 LiveCompare 2.2 release notes

リリース日 2022 年 6 月 14 日

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

Type	Description	ID
Feature	<code>--dry-run</code> 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 <code>.ini</code> ファイルを検証し、いくつかの有用な比較情報を表示できます。詳細については、 ドライランモード を参照してください。	LIV-142, RT78462
Feature	比較セッション中に受信した中止メッセージは時系列で出力されます。	
Feature	テクノロジー、バージョン、接続がタイブレーカーまたは真実の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。	
Feature	<code>'Table Filter'</code> が設定されている場合、表示されます。	
Feature	比較に含まれるテーブルをリストします。これは、 <code>Table Filter</code> を適用した後、すべての接続に存在する共通テーブルのリストです。テーブルごとに、 <code>'Column Filter'</code> 、 <code>'Row Filter'</code> 、および <code>'Comparison Key'</code> が該当する場合。	
Enhancement	x86_64 での SLES 12 のサポート。	LIV-112
Enhancement	x86_64 での SLES 15 のサポート。	LIV-113
Enhancement	Postgres <code>'kwlist'</code> ソースコードに従って、予約語のリストを更新しました。	LIV-141, RT80745
Enhancement	メインの進行状況バーのパーセンテージは、整数の代わりに浮動小数点を使用するようになりました。以前は、数値は切り上げられていました。	LIV-138, RT80745
Enhancement	EPAS で <code>'edb_redwood_date'</code> が有効になっている場合、 <code>'date'</code> 列は <code>'timestamp'</code> 列として作成されます。この型の不一致は、以前に共通ハッシュの不一致として検出され、完全な行比較をトリガーします。これらの場合に <code>'edb_redwood_date'</code> 設定を確認して、共通ハッシュ完全な行より高速でこの不一致を処理することにより、パフォーマンスが向上します。	LIV-153
Enhancement	ハッシュ不一致ログメッセージを WARNING から DEBUG に降格しました。	LIV-145
Enhancement	ログのメッセージに <code>'application_name'</code> が含まれるようになりました。	LIV-141
Enhancement	<code>'data_fetch_mode = prepared_statements'</code> デフォルトの場合、Postgres または EPAS で <code>'Row Filter'</code> の現在のタイムスタンプを使用した動作を文書化しました。この状況では、 <code>'data_fetch_mode = server_side_cursors_with_hold'</code> または <code>'server_side_cursors_without_hold'</code> を設定することも必要です。	LIV-155

次のページに続く

表 6 – 前のページからの続き

Type	Description	ID
Enhancement	検出される発散の数が増加したときに適用 DML スクリプトを生成する際のパフォーマンスが向上しました。	LIV-159
Bug fix	比較キーとして考えられる配列列により、例外が発生してテーブルの比較が中止される問題を修正しました。	LIV-38, LIV-154, RT81758
Bug fix	比較ラウンドの後、ハートビートによってのみ、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が解決されました。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した。比較に時間がかかったテーブルの場合、 <code>min_time_between_heart_beats</code> 、常にゼロを示します。	LIV-149, RT75805
Bug fix	未処理の例外により比較ワーカーがハングする可能性があるコーナーケースを修正しました。	LIV-140, RT80745
Bug fix	Oracle と Postgres を比較する際の誤検出を回避するための 10 進値を正規化します。	LIV-153
Bug fix	<code>comparison_algorithm = block_hash</code> および <code>buffer_size = 1</code> 、および発散が見つかった場合、カーソル進行アルゴリズムの問題のため、比較が進められないコーナーケースを修正しました。	LIV-150
Bug fix	同じテーブルを複数のフィルタセクションで構成できます。	LIV-156

2.6 LiveCompare 2.1 release notes

リリース日 2022 年 3 月 31 日

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

Type	Description	ID
Feature	EDB Postgres 分散 4 のサポート。	LIV-131
Feature	新しい設定 <code>min_time_between_heart_beats</code> は、心拍ごとに比較の進行状況をログに記録するように LiveCompare に指示します。デフォルトでは、 <code>INFO</code> ログレベルを使用して 30 秒に設定されます。	LIV-128

次のページに続く

表 7 – 前のページからの続き

Type	Description	ID
Feature	新しい設定 <code>comparison_cost_limit</code> および <code>comparison_cost_delay</code> 0 より大きい場合、 <code>comparison_cost_limit</code> 行数を処理した後 <code>comparison_cost_delay</code> 秒 0.5 などの仮眠をとるように各ワーカーに指示します。	LIV-16
Change	最近の調査で、一般的なユースケースでパフォーマンスの低下を引き起こすことが判明したため、 <code>parallel_chunk_rows</code> のデフォルト値は 0 に設定され、デフォルトでテーブル分割が無効になります。詳細については、 :ref: 比較モード<比較モード> を参照してください。	LIV-130
Change	<code>CanAdvanceCursors</code> メソッドから処理された行数に関するログメッセージを <code>DEBUG</code> に降格しました。	LIV-129
Bug fix	<code>timestamp(6)</code> データ型の Oracle と Postgres の比較で <code>ORA-01830</code> で失敗する問題を修正しました。	LIV-127
	NA	

2.7 LiveCompare 2.0 release notes

リリース日 2022 年 2 月 15 日

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

Type	Description	ID
Feature	<code>Comparison Key</code> と呼ばれる設定の新しいセクションを使用すると、ユーザーはテーブルごとにカスタム比較キー列のリストを定義できます。これは、主キーまたは一意のインデックスのないテーブルに役立ちます。詳細については、 :ref: 比較キー<比較キー> を参照してください。	LIV-5

次のページに続く

表 8 – 前のページからの続き

Type	Description	ID
Feature	テーブルにユーザー定義の比較キーまたは主キーがない場合、LiveCompare はテーブルの一意のインデックスを使用しようとします。テーブルは複数の一意のインデックスを持つことができるため、LiveCompare は列が Nullable ではないインデックスの使用を優先します。それが不可能な場合、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 NULL 可能列を無視しようとしないうことに注意してください。	LIV-39
Feature	テーブルのすべての列を比較キーとして使用する場合、LiveCompare は NULL 可能列を無視して、すべての列を検討する際の偽陽性を回避するようになりました。この動作は、 <code>`ignore_nullable = false`</code> を設定することにより無効にできます。	LIV-96
Feature	ユーザー定義の比較キーまたはテーブルのすべての列を比較キーとして使用する場合、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。列セットの使用時に重複する値がある場合、LiveCompare はテーブルの比較を中止します。この動作は、 <code>`check_uniqueness_enforcement = false`</code> を設定することにより無効にできます。	LIV-119
Enhancement	Oracle 21c へのサポートを追加しました。	LIV-47
Enhancement	Oracle 12c 以降では、LiveCompare は既に共通のハッシュを使用して <code>`comparison_algorithm = block_hash`</code> および <code>`comparison_algorithm = row_hash`</code> 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ー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 <code>`standard_hash()`</code> ファンクションを使用して Oracle 側で実行されます。LiveCompare では、ユーザーが <code>`sys.dbms_crypto`</code> Oracle システムパッケージに対する <code>`EXECUTE`</code> 特権を持っている場合に、同等のファンクション <code>`sys.dbms_crypto.hash()`</code> を使用して、Oracle 11g で <code>`comparison_algorithm = block_hash`</code> および <code>`comparison_algorithm = row_hash`</code> も許可します。	LIV-79

次のページに続く

表 8 – 前のページからの続き

Type	Description	ID
Enhancement	LiveCompare スキーマをレプリケーション対応 EDB Postgres 分散、pglogical またはネイティブ論理レプリケーションデータベースに追加できるようになりました。	LIV-42
Enhancement	LiveCompare は、PostgreSQL 14 の新しい `pg_read_all_data` ロールを使用できます。	LIV-73
Enhancement	サポートされていないデータベースバージョンの場合、適切なメッセージを表示して中断します。	
Change	パッケージの名前が `2ndq-livecompare` から `edb-livecompare` に変更されました。	
Change	実行可能ファイルの名前が `2ndq-livecompare` から `livecompare` に変更されました。	
Bug fix	Unicode シーケンス `u0000` を適切に引用して、DML 生成時のエラーを回避します。	LIV-98
Bug fix	テーブルが複数の丸い部分に分割されている場合、合計行数が誤って表示される問題を修正しました。	LIV-14
Bug fix	Postgres の空の `bytea` と比較して、Oracle の空の `BLOB` が偽陽性を生成していた問題を修正しました。	LIV-103
Bug fix	接続の問題により、比較セッション全体が中止される例外が発生していた問題を修正しました。これで、LiveCompare は再接続して、可能であれば比較を続行できるようになりました。	LIV-84
Bug fix	発散がある場合の再チェックモードの未処理の例外を修正しました。	LIV-107
Bug fix	接続にテーブルが存在せず、`logical_replication_mode` が無効になっている場合、テーブル比較が中止されない問題を修正しました。	LIV-108
Bug fix	`timestamp` データ型のフィールドが常に Oracle と Postgres 間で一致しないハッシュを生成していた問題を修正しました。	
Bug fix	無視される列が共通ハッシュで検討されたままである問題を修正しました。	

第3章 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+ / EDB Postgres Advanced Server 11+出力接続上
- PostgreSQL / EDB Postgres Extended 9.4+ / EDB Postgres Advanced Server 11+または Oracle 11g+比較するデータ接続で

LiveCompare には、Debian 10+、Ubuntu 16.04+、SLES 12 SP5 および 15 SP3、または CentOS / RHEL/RockyLinux/AlmaLinux 7+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、[EDB customer portal](#) を参照してください。

LiveCompare は、次のいずれかにインストールされます。

- python3 パッケージで提供されている、Ubuntu、Debian、および CentOS / RHEL 8 の最新の Python バージョン
- python-36 パッケージで提供されている CentOS / RHEL 7 の Python 3.6

CentOS / RHEL ディストリビューションでは、LiveCompare には EPEL リポジトリも必要です。詳細については、[EPEL webpage](#) を参照してください。

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

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

Oracle データベースに対して LiveCompare を実行している場合、Oracle Instant Client をインストールする必要があります。詳細については、[Oracle support requirements](#) を参照してください。

3.1 TPAexec を使用した LiveCompare

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

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

(次のページに続く)

```
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: livemlnode1
  location: main
  node: 1
  role: primary
  published_ports:
    - 5401:5432

- Name: livemlnode2
  location: main
  node: 2
  role: replica
  upstream: livemlnode1
  published_ports:
    - 5402:5432
```

TPAexec の詳細については、[EDB customer portal](#) を参照してください。

第4章 Supported technologies

LiveCompare は、PostgreSQL、EDB Postgres DistributedPGD、以前は BDR として知られていた、Oracle を含む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 PGD 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 PGD の場合、および replication_sets または node_name 設定を使用する場合に必要です。logical_replication_mode = pglogical または logical_replication_mode = bdr が必要です。pglogical または PGD 対応のデータベースが必要です。
- **データデータ比較**を実行するためにツールが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、Initial Connection' to gather information about PGD nodes. If logical_replication_mode = bdr and all_bdr_nodes = on, then LiveCompare considers all PGD nodes that are part of the same PGD cluster as the Initial Connection`でも使用されます。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決定は行ごとに機能するため、この修正はすべてのデータ接続に適用できる可能性があります。
- **出力** 必須 LiveCompare は、livecompare というスキーマ、いくつかのテーブル、および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。

この表には、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 でそれらを使用できるコンテキストが示されています。

Technology	Versions	Possible connections
PostgreSQL	10, 11, 12, 13, 14, 15, and 16	Data and output
EDB PostgreSQL Extended (PGE)	10, 11, 12, 13, 14, 15, and 16	Data and output
EDB PostgreSQL Advanced Server (EPAS)	11, 12, 13, 14, 15, and 16	Data and output
pglogical	2 and 3	Initial, data, and output
EDB Postgres Distributed (PGD)	1, 2, 3, 4, and 5	Initial, data, and output
Oracle	11g, 12c, 18c, 19c, and 21c	A single data connection

EDB は、Oracle 10g での LiveCompare のテストを行っていないことに注意してください。そのため、LiveCompare で公式にサポートされている Oracle バージョンのリストには含まれません。Oracle 10g は以前にサポートされており、ほとんどの場合で引き続き動作すると予想されます。ただし、一部の制限が存在する場

合があります。1つの既知の制限は、LiveCompare では `comparison_algorithm` パラメーターを `full_row` `comparison_algorithm = full_row` などに設定する必要があることです。

4.1 PgBouncer のサポート

PgBouncer を介してノードに対して LiveCompare を使用できます。ただし、LiveCompare は PostgreSQL でプリペアドステートメントを使用するため、`pool_mode=session` を使用する必要があります。これは、`pool_mode` が `transaction` または `statement` の場合は不可能です。

第5章 Command-line usage

5.1 比較モード

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。 *Settings* を参照してください。

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

livecompare my_project.ini
```

LiveCompare の実行中、N+1 の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ます。他の進行状況バーは、特定のプロセスによって処理されている現在のテーブルを示します。

各テーブルに表示される情報は、左から右に次のとおりです。

- プロセス数
- テーブル名
- ステータス。現在のテーブルチャンクが後に続く比較ラウンドの ID です。p1/1 は、テーブルが分割されなかったことを意味します。setup のステータスは、テーブルが分析中であることを意味します。行数の確認と必要に応じて分割。
- 処理された行数
- この比較ラウンドで検討される合計行数
- 経過時間
- 完了までの推定時間
- 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 `parallel_chunk_rows > 0`、テーブルに `parallel_chunk_rows` 設定より多くの行がある場合、ハッシュファンクションを使用して、各行を考慮するジョブを決定します。これにより、個々の比較が遅くなる可能性があります。ただし、全体としての比較は、特定のテーブルの並列処理から利益を得る可能性があります。

プログラムの実行中に、**Ctrl-C** を押して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。セッションフォルダー内のマスタープロセスの `PID` この例では `lc\_session\_6` を `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。例

```
livecompare my_project.ini 6
```

プログラムの終了時に、矛盾が検出されなかった場合、出力は次のとおりです。

```
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.
```

不一致が見つかった場合、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:
select * from livecompare.vw_table_summary where session_id = 6;
```

```
Elapsed time: 0:00:50.149987
```

(次のページに続く)

(前のページからの続き)

```
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;
```

5.2 再チェックモード

PGD 環境では、結果的な一貫性のためにレプリケーションが追いついたため、PGD が見つけた相違は後で存在することはできません。いくつかの要因によっては、レプリケーションラグにより、LiveCompare が誤検出を報告する場合があります。

それを克服するために、後でレプリケーションラグが減少するか、データが既に追いついているときに、以前に見つかった差分のみに再チェックを手動で実行できます。この実行モードは *recheck* と呼ばれます。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* または *recheck* で実行を再開する場合、LiveCompare は設定と接続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。発散が見つかった場合、実行を終了し、メッセージを表示します。

5.3 競合モード

conflicts モードで LiveCompare を実行するには、次のコマンドで呼び出します。

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

conflicts モードの詳細については、*PGD support* を参照してください。

5.4 ドライランモード

たとえば、次の INI ファイルがあるとします。

```
[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下の 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.
```

これを修正すると、出力接続に到達できるようになります。ただし、データ接続の 1 つだけが正しく設定されているとします。その場合、LiveCompare は次のエラーで再び失敗します。

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

LiveCompare は、少なくとも 2 つのデータ接続が使用可能な状態で比較を開始できます。そこで、3 番目の接続を修正します。しかし、LiveCompare は次の場合に失敗します。

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

これは、例が `difference_tie_breakers = first` を設定し、タイブレーカーまたは真実のソースとして設定された接続が到達可能であ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、比較を最初からセットアップする場合、LiveCompare が構成エラーで中断するかどうかを事前に確認できます。この種のその他のチェックはすべて、LiveCompare が実行する順序で表示されます。

これは、`--dry-run` モードで行うことができます。

- 構成の問題のために発生するすべての実行中止を出力します。
- 到達可能かどうかを含む、いくつかの詳細を含む接続のリストを出力します。
- テーブルフィルターを印刷します。
- テーブルフィルタを適用した後、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。

次に、`.ini` ファイルの例と、到達不能な接続に関するすべての構成エラーを指定した 1 つのサンプル出力を示します。

```
$ 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 | PGD Version | Pglogical Version | Initial |
| Tie Breaker | Source of Truth | Reachable |
+-----+-----+-----+-----+-----+-----+
| second | postgresql | 110015 | - | - | False |
| False | False | True |
| first | postgresql | - | - | - | False |
| True | False | False |
| third | postgresql | - | - | - | False |
| False | False | False |
| output | postgresql | - | - | - | - |
| - | - | False |
```

(次のページに続く)

(前のページからの続き)

```
+-----+-----+-----+-----+
→+-----+-----+-----+-----+

Table Filter

publications =
replication_sets =
schemas = schema_name = public
tables =

Tables

+-----+-----+-----+-----+
| Table Name          | Row Filter  | Column Filter | Custom Comparison Key |
+-----+-----+-----+-----+
| public.categories   | -           | -             | -                       |
| public.cust_hist     | -           | -             | -                       |
| public.customers     | -           | -             | -                       |
| public.departments   | -           | -             | -                       |
| public.dept_emp      | -           | -             | -                       |
| public.dept_manager  | -           | -             | -                       |
| public.employees     | -           | -             | -                       |
| public.inventory     | -           | -             | -                       |
| public.orderlines    | -           | -             | -                       |
| public.orders        | -           | -             | -                       |
| public.products     | -           | -             | -                       |
| public.reorder       | -           | -             | -                       |
| public.salaries      | -           | -             | -                       |
| public.tbl           | -           | -             | -                       |
| public.titles        | -           | -             | -                       |
+-----+-----+-----+-----+
```

第6章 Advanced usage

LiveCompare が実行されると、作業ディレクトリに `lc_session_<session_id>` という名前のフォルダーが作成されます。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。テーブルごとに、LiveCompare がテーブルの処理に要した時間、合計行数と処理された行数、テーブルで見つかった相違の数、および無視される列の最大数存在する場合を示します。

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` —違いに関する役立つ情報。違いがない場合、このファイルは生成されません。

以下は差分リストの例です。

```
+-----+-----+-----+-----+
↪+
| 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              |
↪|
+-----+-----+-----+-----+
↪+
```

すべての詳細と差分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決定するためにどのように機能したかを理解するために、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、最初のデータベースに適用して、他のすべてのデータベースと一貫性を保つ DML コマンドを示します。以下は、表に示されている相違点のスキプトの例です。

```
BEGIN;

DELETE FROM public.categories WHERE (category) = 7;
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 はこのスキプトを生成します。最初のデータベースの不一致を修正するには、そのデータベースでスキプトを実行します。

LiveCompare は、一貫性のないデータがあるデータベースごとに同様の `apply_on_*.sql` スキプトを生成します。

6.1 比較の中止

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。到達可能な接続の数が少なくとも 2 つではない場合、LiveCompare はセッション全体を中止し、エラーメッセージを表示します。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。すべての接続で、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対して、LiveCompare はデータベーステクノロジーと `logical_replication_mode` 設定の周りのサニティチェックを実行します。サニティチェックのいずれかが失敗すると、LiveCompare は比較を中止し、エラーメッセージを表示します。

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、テーブルフィルタを考慮して、比較するテーブルのリストを構築します。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。この手順は**セットアップ**と呼ばれます。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へのアクセスがない場合、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を表示して終了します。それ以外の場合、エラーが発生したテーブルのみがテーブル比較を中止し、LiveCompare は次のテーブルに移動します。

LiveCompare がテーブル比較を開始するテーブルごとに、LiveCompare は最初に到達可能なすべての接続でテーブル定義をチェックします。テーブルの列と列のデータ型が同じでない場合、LiveCompare は column_intersection を適用します。比較する列がない場合、LiveCompare はテーブル比較を中止します。

6.2 比較キー

比較されるテーブルごとに、テーブルのメタデータを収集するときに、LiveCompare は、次のルールに従って、テーブル比較で使用する比較キーを構築します。

1. 設定されている場合、カスタム比較キーを使用します。
2. または、可能な場合は PK を使用します。
3. または、表に UNIQUE インデックスがある場合、すべての NOT NULL 列を持つ UNIQUE インデックスのうち、列の少ない UNIQUE インデックスを使用します。
4. これら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ます。ignore_nullable = false の場合、NULL 列も考慮されます。

戦略 1 または 4 を比較キーとして使用することにした場合、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 はそのテーブルの比較を中止します。check_uniqueness_enforcement = false を使用して、この動作を無効にできます。

6.3 修正する違い

LiveCompare は、次の違いを特定して修正を提供できます。

- 行は大部分のデータ接続に存在します。修正は、多様なデータベースの INSERT です。
- 大部分のデータ接続には行が存在しません。修正は、発散データベースの DELETE です。
- すべてのデータベースに行が存在しますが、一部の列値が不一致です。修正は、多様なデータベースの UPDATE です。

デフォルト設定は difference_statements = all で、これは、LiveCompare が、検出した差分ごとに3つの DML タイプ INSERT、UPDATE、および DELETE をすべてを適用しようとすることを意味します。ただし、差分修正を提供するときに考慮する LiveCompare の DML のタイプを指定できます。difference_statements 設定の値を次の値のいずれかに変更します。

- all デフォルト INSERT、UPDATE、および DELETE DML タイプを修正します。
- inserts INSERT DML タイプのみを修正します。

- updates UPDATE DML タイプのみを修正します。
- deletes DELETE DML タイプのみを修正します。
- inserts_updates INSERT および UPDATE DML タイプのみを修正します。
- inserts_deletes INSERT および DELETE DML タイプのみを修正します。
- updates_deletes UPDATE および DELETE DML タイプのみを修正します。

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。

6.4 差分ログ

テーブル difference_log には、LiveCompare が差分をチェックするたびに、差分に関するすべての情報を保存します。LiveCompare は再チェックモードで複数回実行できるため、この表は、LiveCompare が再チェックしていた時間枠で差がどのように変化したかを示しています。

- **Detected (D)** 差分が検出されたばかりです。再チェックおよび修正モードでは、LiveCompare はすべての永続的な差分とタイの差分を検出済みとしてマークするため、それらを再チェックできます。
- **Permanent (P)** 差分を再確認した後、データがまだ発散している場合、LiveCompare は差分を Permanent としてマークします。
- **Tie (T)** このエントリーはパーマネントと同じですが、マジョリティである接続を決定するための十分なコンセンサスがありません。
- **Absent (A)** 再チェック時に、LiveCompare で差分が存在しないことが判明した場合、つまり、行が両方のデータベース間で一貫している場合、LiveCompare は差分を Absent としてマークします。
- **Volatile 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 はその差を Volatile としてマークします。
- **Ignored(I)** 出力 PostgreSQL 接続でファンクション<livecompare_schema_name>.accept_divergence(session_id, table_name, difference_pk) を手動で呼び出すことにより、特定の差異の差分再チェックを停止できます。例

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

第7章 EDB Postgres Distributed support

LiveCompare は、EDB Postgres DistributedPGD、以前は BDR ノードと同様に非 PGD ノードに対して使用できます。

`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 PGD クラスターに属していると想定します。次に、ノード名をフィルタテーブルへの接続およびレプリケーションセットとして指定できます。

たとえば、PGD クラスター内の任意のノードに接続できるとします。これを初期接続と呼びます。最初にこのノードに接続することにより、LiveCompare は PGD 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 PGD ノードを比較するとします。LiveCompare は初期接続から任意のノードに接続できるため、データ接続の `dsn` または接続情報を定義する必要はありません。 `node_name` を定義するだけで済みます。LiveCompare は、そのノードの接続情報に関する PGD メタデータを検索し、ノードに接続します。

LiveCompare が PGD メタデータを取得することにより他のすべてのノードに接続するには、LiveCompare がフィールド `interface_connstr` の PGD ビュー `bdr.node_summary` から同じ DSN を使用してそれらに接続する必要があります。この場合、postgres ユーザーとしての初期接続と同じマシンで 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` 属性を定義します。

レプリケーションセットをテーブルフィルタとして指定することもできます。LiveCompare は、PGD メタデータを使用してテーブルリストを構築し、`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検討します。

たとえば、`.ini` ファイルを作成して、3 つの PGD 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
```

(次のページに続く)

(前のページからの続き)

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

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

PGD 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

1. General Settings の下で、all_bdr_nodes = on を有効にします。
2. Initial Connection の下で、初期接続を指定します。

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]
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 設定を使用してすべての PGD 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が、設定すると、データ接続のリストに追加されます。たとえば、PGD クラスター全体を単一の Postgres 接続と比較できます。これは、移行プロジェクトに役立ちます。

```
[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
```

(次のページに続く)

(前のページからの続き)

[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。

- PGD 1、2、3、および 4
- pglogical 2 および 3

PGD の代わりに pg ロジカルメタデータの取得を有効にするには、`logical_replication_mode = bdr` の代わりに `logical_replication_mode = pglogical` を設定します。

7.1 PGD 監視ノード

PGD のレプリケーションセットを使用して、PGD レプリケーションに含める特定のテーブルを構成できます。テーブルが属してい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、これらのテーブルからデータを受信するノードを指定することもできます。この設定では、PGD シャーディングや PGD 監視ノードの使用などのさまざまなアーキテクチャが許可されます。

PGD 監視は、他のノードから DML をレプリケートしない通常の PGD 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。レプリケーションセットの構成は、監視が DDL をレプリケートするかどうかを決定します。これは、2 種類の PGD 証人が存在することを意味します。

- データもテーブルも含まない完全に空のノード
- 他のノードから DDL を複製するノードのため、空のテーブルがあります

最初のケースでは、PGD 監視が比較に含まれている場合でも `[Connections]` で手動でまたは `all_bdr_nodes = on` を使用して、`witness` に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
```

2 番目のケースでは、テーブルは PGD 監視に存在します。ただし、証人で不足しているデータを発散として報告するのは正しくありません。したがって、LiveCompare は、テーブルごとに、比較に含まれる各ノードに関する次の情報をチェックします。

- 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
- Table Filter の下のフィルタ `replication_sets` で定義したレプリケーションセットがある場合

レプリケーションセットの 3 つのリストすべての交差が空の場合、PGD 監視の場合、LiveCompare は次のメッセージ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レプリケートされます。

7.2 PGD クラスターの違い

LiveCompare は、ローカルノードにのみ変更を加えます。修正的な変更は他のノードに複製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` というレプリケーションオリジンが既に存在するかどうかを確認します。`difference_fix_replication_origin` 設定を調整することにより、レプリケーションオリジンの名前を構成できます。

重要: PGD 3.6.18 では、ローカル専用トランザクションを適用するために使用する新しい既存の ``bdr_local_only_origin`` レプリケーションオリジンを導入しました。LiveCompare が PGD 3.6.18 に接続されている場合、このレプリケーションオリジンは作成されません。

LiveCompare は、次を考慮して適用スクリプトを生成します。

- レプリケーションオリジン `bdr_local_only_origin` を使用するように現在のトランザクションを設定するため、実行される DML には `bdr_local_only_origin` に関連付けられた `xmin` があります。
- 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 PGD 競合がある場合、LiveCompare DML は常に競合に負けます。

LiveCompare 修正スクリプトを PGD ノードに適用した後、次のクエリーを使用して 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ません。

LiveCompare が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。

レプリケーションオリジンに関係するこれらの手順はすべて、PostgreSQL ユーザーが `bdr_superuser` または PostgreSQL スーパーユーザー権限を持っている場合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に関連付けずに修正を生成します。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。それ以外の場合、LiveCompare は、競合の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を持ちます。また、修正に関連付けられているレプリケーションオリジンが存在しないため、クエリを使用して LiveCompare によって修正されたすべての行をリストすることはできません。

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

- `bdr.difference_fix_origin_create()` このレプリケーションオリジンが存在しない場合、LiveCompare によって実行され、`difference_fix_replication_origin` デフォルトでは `bdr_local_only_origin` に設定で指定されたレプリケーションオリジンを作成します。
- `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` で指定されたレプリケーションオリジン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、トランザクションははるかに過去の `2010-01-01` に設定されます。LiveCompare によって適用された修正トランザクションは、常に競合を失います。

これらの機能には、PostgreSQL スーパーユーザーではなく `bdr_superuser` が必要です。PGD 3.7.0 以降、これらの機能は非推奨です。その場合、PostgreSQL スーパーユーザーとして実行している場合、LiveCompare は次の機能を使用して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もののみを含めます。

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

7.3 PGD の競合

LiveCompare には、`conflicts` と呼ばれる実行モードがあります。この実行モードは、PGD クラスターに固有です。PGD 3.6、PGD 3.7、PGD 4、および PGD 5 クラスターでのみ動作します。

`compare` モードは、テーブルのすべてのコンテンツを全体的に比較するために使用されますが、`conflicts` モードは、PGD 3.6 の場合は `bdr.apply_log`、または PGD 3.7 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当てます。、PGD 4、および PGD 5。

`conflicts` 実行モードは、特定のテーブルから特定のタプルのみを検査するため、`compare` モードよりもはるかに高速に実行されることが予想されます。ただし、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、PGD によって実行されている自動競合解決がノード間で一貫していることを確認することです。つまり、PGD が競合を解決した後、クラスターは一貫した状態になります。

一般的なユースケースでは、自動競合解決によりクラスターの一貫性が保証されますが、自動競合解決がノード間で分岐したタプルを生成する可能性がある既知の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、一貫性のチェックと保証に役立ち、時間と結果の適切なバランスを提供します。

7.3.1 競合の例

`node3` で、次のクエリー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
FROM bdr.conflict_history c
INNER JOIN bdr.subscription_summary s
ON s.sub_id = c.sub_id;
```

`bdr.conflict_history` では、次の競合が確認できます。

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

この競合は、`DELETE` が `node2` から `node3` に到着したときに、テーブル `tbl` に `b = 3` を含む行がなかったことを意味します。ただし、`INSERT` は、後で `node1` から `node3` に到着した可能性があり、`b = 3` を含む行を `node3` に追加します。したがって、これが `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` では、これが表示されます。

```
bdrdb=# SELECT * FROM tbl WHERE b = 3;
 a | b | c
```

(次のページに続く)

(前のページからの続き)

```
- -+-----+--
(0 rows)
```

PGD 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、`compare` モードで LiveCompare を実行できます。ただし、比較セットのサイズによっては、テーブル `tbl` が非常に大きい場合、これには長時間、場合によっては数時間かかる場合があります。

この状況は、`conflicts` モードが役立つ場合です。この場合、`delete_missing` 競合は `node3` からのみ可視されますが、LiveCompare は競合ログ行 `key_tuple`、`local_tuple`、`remote_tuple`、および `apply_tuple` から PK 値を抽出し、影響を受けるテーブルでのみクラスター全体の自動比較を実行できます。PK 値によるフィルタリング。比較では、クラスター内のすべてのノードの現在の行バージョンをチェックします。

`all_bdr_nodes = on` を設定する、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 に指示するための `check.ini` ファイルを作成します。

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

[Initial Connection]
dsn = dbname=bdrdb

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

`conflicts` モードで LiveCompare を実行するには

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

実行後、コンソール出力に次のようなものが表示されます。

```
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/` `X` は、現在の比較セッションの番号です。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ます。ファイルには、すべての発散的な競合に関する主な情報が表示されます。

データベース `liveoutput` に接続すると、たとえば次のクエリーを使用して、競合の詳細を表示できます。

```
SELECT *
FROM livecompare.vw_conflicts
```

(次のページに続く)

(前のページからの続き)

```
WHERE session_id = 1
      AND conflict_id = 1
ORDER BY table_name,
         local_time,
         target_node;
```

出力は次のようなものです。

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
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現在発散しているノードを見つけたことを意味します。livecompare.vw_conflicts ビューは、非発散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 現在の日のファイル名の DAY も生成します。

```
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;
```

LiveCompare は、他の 2 つの行に存在しないため、node3 から b = 3 が存在する行を DELETE に提案しています。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と、PGD クラスターの一貫性が再び高まります。

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

--conflicts モード比較は、完全な--compare よりもはるかに高速であるため、--conflicts 比較セッションをより頻繁にスケジュールして、競合解決がクラスター全体の一貫性を提供していることを確認することを強くお勧めします。

注釈: PGD 3.7 の `bdr.conflict\_history` または PGD 3.6 の `bdr.apply\_log` のデータを表示するには、bdr_superuser または PostgreSQL スーパーユーザーであるユーザーで LiveCompare を実行します。

PGD 3.7+の bdr.conflict_history または PGD 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あるユーザーで LiveCompare を実行します。

7.3.2 競合フィルター

bdr.conflicts_history または bdr.apply_log の列で競合をフィルターするように 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
```

7.4 ミキシングテクノロジー

node_name および replication_sets のメタデータは、初期接続で取得されます。したがって、pglogical および/または PGD 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構築されます。したがって、最初の接続で replication_sets 条件が有効である必要があります。

混合テクノロジーの比較を実行できます。例

- PGD 1 ノードと PGD 3 ノード
- PGD 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 pg 論理ノード

第8章 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 PGD 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。次に、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。

次に、Oracle データベースと PostgreSQL データベースの単純な比較の例を示します。

```
[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

[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]
```

(次のページに続く)

```
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>`
- Oracle `<user> . <table_name>`

`schema_qualified_table_names = off` を設定することにより、スキーマ修飾テーブル名を無効にできます。これは、`oracle_user_tables_only = on` の場合にのみ実行できます。この設定は、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ように LiveCompare に指示します。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、ロールまたは構成でデフォルトの `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` を設定できます。たとえば、

[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
```

[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 で非修飾テーブル名を使用することもできます。

注釈: *Output Connection* は、LiveCompare から進行状況とレポート情報を書き込むために必要です。

PGD データベースを Oracle と比較する必要がある、Initial Connection、node_name、および replication_sets 機能 *PGD support* で説明を利用したい場合、次のように最後のデータ接続を Oracle にポイントできます。

```
[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
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
```

PGD クラスター全体を単一の Oracle データベースと比較することもできます。たとえば、

```
[General Settings]
logical_replication_mode = bdr
max_parallel_workers = 4
oracle_user_tables_only = on
oracle_ignore_unsortable = on
```

(次のページに続く)

```
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
```

8.1 要件

LiveCompare は、そのまま PostgreSQL データベースで動作します。追加のソフトウェアをインストールする必要はありません。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。

8.1.1 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か、使用す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する必要があります。

- **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、LD_LIBRARY_PATH を設定します。
- **Windows** Oracle Instant Client **32-bit 64-bit** をダウンロードし、LiveCompare フォルダーに抽出します。

8.1.2 cx_Oracle Python モジュール

LiveCompare が Oracle データベースに接続できるように、Python モジュール

`cx_Oracle` をインストールしてシステムに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リポジトリからインストールできないため、

the instructions on the `cx_Oracle` website に従ってシステムにインストールします。

postgres オペレーティングシステムユーザーで LiveCompare を実行することをお勧めします。次に、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
```

8.2 の違い

LiveCompare は違いを見つけた場合、PostgreSQL 接続にのみ適用する DML スクリプトを生成します。Oracle 接続に適用する DML スクリプトは生成されません。

8.3 BLOB および CLOB データ型

PostgreSQL の同等のフィールドが `text` 型である場合、LiveCompare は Oracle の CLOB フィールドを比較できます。Oracle の BLOB フィールドにも同じことが当てはまります。PostgreSQL の同等のものは `bytea` 型です。

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、BLOB / CLOB フィールドを処理しません。その場合、テーブルは無視され、LiveCompare ログには次のようなメッセージが記録されます。

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

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、この動作をワークアラウンドできます。General Settings セクションで次の 2 つの設定を有効にします。

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

8.4 互換性のない照合順序

Oracle では、一般的に次の初期化パラメーターが設定されます。

```
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順序は C 照合順序です。Postgres データベースが別の照合順序で初期化された場合、デフォルトでは、LiveCompare は PK 値をソートするときに問題を検出する場合があります。これにより、誤検出が発生する可能性があります。

これをワークアラウンドするには、Postgres で照合順序たとえば C 照合順序を強制して、Oracle と同じソート動作と一致させることができます。

```
force_collate = C
```

比較セッションに Oracle および PostgreSQL が関係することを LiveCompare が検出した場合、LiveCompare は別の値に設定しない限り `force_collate = C` を既に設定しています。

8.5 共通ハッシュ

デフォルトでは、PostgreSQL と Oracle を比較する場合でも、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、行はテキストとして完全に表されます。
- Postgres 側では、タイムスタンプ、数値、および bytea データ型が Oracle を模倣するように処理されます。
- 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` に設定して使用できます。
- `block_hash` の使用時に不一致があった場合、Postgres と Postgres の比較の場合と同様に、LiveCompare は `row_hash` に、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 文字にのみ制限されています。`comparison_algorithm = full_row` では、BLOB および CLOB 全体の比較ができます。
- Oracle では、完全な行表現の幅が 4000 文字以下である必要があります。完全な行表現が 4000 文字を超えている場合、LiveCompare はその特定のテーブルの比較を中止し、次のエラーメッセージがログに追加されます。

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

以降の LiveCompare バージョンでは、これらの特定のテーブルでの `full_row` 比較にフォールバックします。今のところ、回避策は、`comparison_algorithm = full_row` を使用して、これらのテーブルでのみ個別の比

較セッションを構成することです。Oracle 10g で LiveCompare を使用するには、常に `comparison_algorithm = full_row` を設定する必要があります。

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ます。ただし、Oracle 側のユーザーの追加の特権が必要になる場合があります。例

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


第9章 Settings

9.1 一般設定

- `logical_replication_mode` プログラムが接続とテーブルフィルタ設定を解釈する方法、および比較を開始する前に接続でチェックする要件に影響します。現在、使用可能な値は次のとおりです。
 - `off` データベース間に論理レプリケーションがないことを想定しています。
 - `native` データベース間にネイティブ論理レプリケーションがあることを想定しています。Table Filter -> `publications` 設定の使用を有効にして、使用するテーブルのリストを指定します。すべてのデータベースで PostgreSQL 10+が必要です。
 - `pglogical` データベース間に pg 論理レプリケーションがあることを前提としています。Table Filter -> `replication_sets` 設定の使用を有効にして、使用するテーブルのリストを指定します。また、`node_name` を使用してデータ接続を指定できるようにします。これには、ノードの DSN 情報を取得するために使用される Initial Connection の設定が必要です。すべてのデータベースに pglogical 拡張機能をインストールする必要があります。
 - `bdr` すべてのデータ接続が同じ PGD クラスターからのノードであると仮定します。Table Filter -> `replication_sets` 設定の使用を有効にして、使用するテーブルのリストを指定します。また、`node_name` を使用してデータ接続を指定できるようにします。これには、ノードの DSN 情報を取得するために使用される Initial Connection の設定が必要です。すべてのデータベースに pglogical および bdr 拡張機能をインストールする必要があります。
- `all_bdr_nodes logical_replication_mode` が `bdr` に設定されている場合、初期接続のみを指定し、LiveCompare にアクティブな PGD 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` 考慮する並列プロセスの数。各プロセスは、キューから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 のオープン接続を保持します。1 つは各データ接続に、もう 1 つは出力データベースにです。

- `buffer_size` すべてのデータ取得操作でテーブルから取得する行数。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` データの取得にプリペアドステートメント `LIMIT` を使用したクエリーを使用します。毎回、非常に少量のデータデフォルトでは `buffer_size = 4096` 行のみが取得されるため、3つのモードすべての中で最も影響が最も小さく、同じ理由でより安全な取得モードです。非同期データの取得を許可します `parallel_data_fetch` で定義一般的なユースケースでは、この取得方法は良好なパフォーマンスを提供しますが、大規模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、サーバー負荷が中～高の場合に強くお勧めします。
- `server_side_cursors_with_hold` データの取得にサーバーサイドのカーソル `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、肥大化とレプリケーションの問題が発生し、`VACUUM` の正常な実行を妨げる可能性があります。また、`WITH HOLD` 句は、クエリをマテリアライズするように Postgres に指示するため、ワーカーはデータがマテリアライズされるまで数秒ハングする可能性がありますため、テーブルデータ全体が RAM を消費し、一時ファイルとして Postgres サイドディスクに保存できます。 `parallel_chunk_rows` デフォルトで無効を使用することにより、その影響をすべて軽減し、`buffer_size` を少し増やすことにより速度を向上させることができます。非同期データの取得を許可します `parallel_data_fetch` で定義一般的な使用ケースでは、この取得方法は `prepared_statements` と比較して利点はありませんが、複数の小さなテーブルの場合、より高速です。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` データの取得にサーバーサイドのカーソル `WITHOUT HOLD` を使用します。 `server_side_cursors_with_hold` と同様、このモードは `xmin` を抑制することもできるため、Postgres での肥大化、`VACUUM`、およびレプリケーションの問題を発生させる可能性があります。ただし、`WITHOUT HOLD` カーソルは比較セッション全体でオープンなトランザクションを必要とするため、このような影響はより高くなります。この要件は後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で保持されるため、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。クエリは実体化されないため、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、大規模なテーブルの場合、このモードの方が速くなる場合があります。データベースの負荷が低～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参照できるということですが、これは、ビジー状態のデータベースでサーバーサイドのカーソルを使用している場合には発生しない場合があります。したがって、実稼働の高負荷サーバーには `prepared_statements` を使用し、テスト、移行シナリオ、低負荷サーバーには `server_side_cursors_*` 設定のいずれかを使用することをお勧めします。最良の戦略は、おそらく、非常に大規模なテーブルに `server_side_cursors_without_hold` を使用し、残りのテーブルに `prepared_statements` を混合することです。次の表は、費用/便益比の比較を示しています。

	prepared ments	state- server side cursors with hold	server side cursors without hold
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

- **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最小行数。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。各テーブルチャンクには **parallel_chunk_rows** 行以下があります。任意の値<1 に設定すると、テーブル分割が無効になります。デフォルト 0 無効。

重要: テーブル分割は、複数のワーカーが大きなテーブルを並列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 の構成によっては、特にデフォルトの `random\_page\_cost = 4` では、最近のハードドライブには保守的すぎると考えられますが、Postgres クエリプランナーは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 でビットマップヒープスキャンを無効にすると、比較パフォーマンスが大幅に向上します。 *start_query* 設定を使用して、接続ごとにこれを行うことができます。

```
start_query = set enable_bitmapsan = off
```

- **parallel_data_fetch** データの取得を並列的に実行するかどうかを指定しますつまり、データベースへの非同期接続を使用します。多方向比較のパフォーマンスが向上します。PostgreSQL ではない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。ハッシュを使用すると、完全な行比較よりも高速です。次の値のいずれかになります。
 - **full_row** ハッシュを使用した行比較を無効にします。この場合、完全な比較は、行を列ごとに比較することにより実行されます。Oracle 10g データベースを含む比較の場合、 **full_row** は唯一の有効な **comparison_algorithm** 設定値であることに注意してください。
 - **row_hash** ハッシュを使用した行比較を有効にし、テーブルの分割を有効にします。テーブルは分割されているため、各ワーカーはテーブルごとの最大 **parallel_chunk_rows** を比較します。PostgreSQL ではデータ行がハッシュされるため、比較は **full_row** よりも高速です。ただし、特定の行のハッシュが一致しない場合、その特定の行に対して、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。データ接続が PostgreSQL ではない場合、LiveCompare は、検討中の行の連結列値の MD5 ハッシュとして定義された行ハッシュ、比較対象のデータベーステクノロジー間の**共通ハッシュ**を使用します。
 - **block_hash** 同様に動作します **row_hash** と同様ですが、LiveCompare は行ごとに比較する代わ

りに、**ブロックハッシュ**、つまり、取得されたばかりのデータバッファー最大 `buffer_size` 行のすべての行のハッシュのハッシュをビルドします。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。ブロックの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` よりも高速である理由です。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、バッファーで行ごとに比較を実行して、発散する行を見つけます。これはデフォルト値です。

- `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秒単位の時間。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` 比較アルゴリズムが進行中のときに各ラウンドの状態を更新するまで待機する秒単位の時間。ラウンド保存はハートビート中にのみ発生できるため、`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の状態を更新します。デフォルト 60 秒。

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

- `comparison_cost_limit > 0` の場合、`comparison_cost_delay` 秒の昼寝をする前に各ワーカーが処理する行数に対応します。デフォルト 0 は、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のスリープ時間を指定します。デフォルト 0.0
- `stop_after_time` **Ctrl-C** を押したかのように LiveCompare が停止する秒単位の時間。コマンドラインの引数としてセッション ID を渡すことにより、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再開できます。デフォルト `stop_after_time = 0`、これは自動中断が無効であ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。使用可能な値は `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 `simple_majority`
- `difference_required_quorum consensus_mode = quorum_based` の場合、この設定は、どの接続が発散してい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であるため、使用をお勧めしません。デフォルト値は 0.5 であり、それに近い値を使用することをお勧めします。
- `difference_sources_of_truth` 真実のソースとして検討する接続名またはノード名、`logical_replication_mode = bdr` および `all_bdr_nodes = on` の場合、ノード名をコンマ区切りのリスト。`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、`node1` Connection および `node2` Connection のいずれかを `.ini` ファイルまたは `all_bdr_nodes = on` で定義する必要があり、Initial Connection のみが定義されますが、`node1` および `node2` は有効な PGD 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状況を見つけたときは常に、タイブレーカーとして考えられる接続名またはノード名、**logical_replication_mode = bdr** および **all_bdr_nodes = on** の場合はノード名。例 **difference_tie_breakers = node1,node2** この例では、セクション **node1 Connection** および **node2 Connections** のいずれかを **.ini** ファイルまたは **all_bdr_nodes = on** で定義する必要があり、**Initial Connection** のみが定義されますが、**node1** および **node2** は有効な PGD ノード名である必要があります。デフォルト接続をタイブレーカーとして考慮しません。
- **difference_statements** LiveCompare が生成する 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 デフォルト
 - inserts
 - updates
 - deletes
 - inserts_updates
 - inserts_deletes edb_17_
 -
- **difference_allow_null_updates** **UPDATE SET col = NULL** のような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 が生成する 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 デフォルト
 - insert_update_delete
 - insert_delete_update
 - update_insert_delete
 - update_delete_insert
- **difference_fix_replication_origin** PGD 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オリジンがまだ存在しない場合は作成します。次に、レプリケーションオリジンを使用して、DML 修正を含む適用スクリプトを作成します。**difference_fix_replication_origin** は、LiveCompare が使用するレプリケーションオリジンの名前を指定します。この設定の値を設定しない場合、LiveCompare は **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、比較後の検証を可能にするためにドロップされません。ただし、必要に応じて、後でレプリケーションオリジンを手動でドロップできます。**logical_replication_mode = bdr** が必要です。

重要: PGD 3.6.18 では、ローカル専用トランザクションを適用するために使用する新しい事前作成された ``bdr_local_only_origin`` レプリケーションオリジンを導入しました。したがって、LiveCompare が PGD

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、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 によって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ー。さらに、PGD 比較が実行されており、**difference_fix_start_query** が空の場合、LiveCompare は以下も自動的に行います。
 - 発散接続が PGD 3.6.7 の場合、`SET LOCAL bdr.xact_replication = off;`
 - **difference_fix_replication_origin** で指定されたレプリケーションオリジンを使用するようにトランザクションを設定するコマンドを追加します。
- **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、比較レポートテーブルが作成されるスキーマ。デフォルト `livecompare`
- **hash_column_name** すべてのデータ取得には、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** 一部の取得では、クエリ列内で `row_number()` ファンクション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** この設定が有効になっている場合、LiveCompare は発散する各行のオリジン名を取得します。これは、デバッグの目的で役立つ場合があります。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。デフォルト `off`
- **column_intersection** この設定が有効になっている場合、比較される特定のテーブルについて、LiveCompare はすべての接続でテーブルの列の交差点でのみ動作し、接続に存在する可能性のある追加の列を無視します。この設定が無効になっている場合、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、列の不一致がある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で、LiveCompare が主キーとは異なる比較キーを使用している場合、LiveCompare は **ignore_nullable** が有効になっているデフォルトの場合、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できます。これにより、LiveCompare は比較で Null 許容列を考慮できます。これは、一部のコーナーケースで偽陽性を生成する可能性があります。
- **check_uniqueness_enforcement** LiveCompare がユーザー定義の比較キーを使用しているか、テーブル内のすべての列を比較キーとして使用している場合、LiveCompare は **check_uniqueness_enforcement** 設定が有効になっているデフォルトで比較キーの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 のソート不可データ型 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、**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 にログインした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。このアプローチは、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、高速です。デフォルト `off`
- `oracle_fetch_fk_metadata` 有効にすると、外部キーメタデータを取得するように LiveCompare に指示します。これは、遅い操作に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` この設定が有効になっている場合、テーブル名はスキーマ修飾されたものとして扱われます。これを無効にすると、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 の比較では、`oracle_user_tables_only` も有効にする必要があります。Postgres x Postgres では、同じデータベース内でも、異なるスキーマの下にあるテーブルの比較ができます。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、スキーマ名のないテーブル名を許可します。デフォルト `on`
- `force_collate` `off` 以外の値および有効な照合名に設定されている場合、比較対象のすべての 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。異なる照合順序で 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、`force_collate = C` を設定します。混合テクノロジー Oracle と PostgreSQL などと比較し、照合順序が指定されていない場合、値 `C` になります。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この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 がエラーを発生すると、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` 設定は ``compare`` モード中にのみ考慮されます。 `recheck` モードでは、LiveCompare は常にセットアップの最初のエラーで強制終了します。

- `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。デフォルトの動作では、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。定数のみを設定する必要があり、定数の周囲の `$` シンボルは設定する必要はありません。デフォルト `off`、これは LiveCompare がワード 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、発散があるノードで 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したい場合に役立ちます。これを有効にするには、PostgreSQL のスーパーユーザーが必要です。それ以外の場合、効果はありません。デフォルト `off`
- `split_updates` 有効にすると、LiveCompare は UPDATE 発散を分割します。つまり、UPDATE DML を生成する代わりに、適用スクリプトで対応する DELETE および INSERT を生成します。デフォルト `off`

- `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 が丸める 10 進数を指定する整数。デフォルト -1、浮動小数点の丸めを無効にします。

9.2 初期接続

初期接続は、`logical_replication_mode` が `pglogical` または `bdr` に設定されている場合にのみ使用されます。`node_name` 設定のみを使用するようにデータ接続を設定している場合、プログラムがノード名から DSN の取得を開始するときに使用されます。

- `technology` RDBMS テクノロジー。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場合は、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` データベースユーザー。デフォルト `postgres`
- `application_name`. アプリケーション名。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`

9.3 出力接続

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

- `technology` RDBMS テクノロジー。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場合は、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` データベースユーザー。デフォルト `postgres`
- `application_name`. アプリケーション名。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`

9.4 データ接続

データ接続は、Initial Connection および Output Connection と同様の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取得および比較します。

Initial Connection および Output Connection と同様に、データ接続は名前付けセクションで定義されます。セクション名は<Name> Connection の形式であり、<Name> は、英文字で始まる任意の 1 ワード文字列です。この場合、Name として使用するものは、データ接続の **接続 ID** と呼ばれます。各データ接続には、データ接続のリスト内で一意の接続 ID が必要です。

logical_replication_mode = bdr および all_bdr_nodes = on の場合、データ接続を指定する必要はありません。LiveCompare は、Initial Connection から PGD メタデータを取得することにより、データ接続リストを構築します。

- **technology** RDBMS テクノロジー。現在使用可能な値は 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、Initial Connection が埋められることも必要です。node_name が設定されている場合、dsn、host、port、dbname、および user 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場合は、空のままにします。
- **port** ポート。デフォルト 5432
- **dbname** データベース名。デフォルト postgres
- **service** Oracle 接続で使用するサービス名。デフォルト XE
- **user** データベースユーザー。デフォルト postgres
- **password** プレーンテキストのパスワード。これの使用はお勧めしません。ただし、一部の古い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。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くたびに実行される任意のクエリー。
- **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かどうかを指定します。デフォルト on

9.5 テーブルフィルタ

.ini ファイルのこのセクションは、省略するか空のままにした場合、最初のデータベースのすべてのテーブルに対して LiveCompare が実行され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ます。LiveCompare は、これらのパブリケーションに関連付けられているテーブルのみを使用します。変数 **publication_name** を使用して、条件式を構築できます。たとえば、

```
publications = publication_name = livepub
```

logical_replication_mode = native が必要です。

- **replication_sets** pglogical または PGD を使用する場合、特定のレプリケーションセットをフィルタリングでき、LiveCompare はこれらのレプリケーションセットに関連付けられているテーブルでのみ動作します。変数 **set_name** を使用して、条件式を構築できます。たとえば、

```
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動作します。変数 **schema_name** を使用して、条件式を構築できます。たとえば、

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

- **tables** 変数 **table_name** は、LiveCompare で作業するテーブルのみをフィルタリングする条件式を構築するのに役立ちます。例

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

条件式で、% 文字を%% としてエスケープします。

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リングできます。

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

一般設定 **schema_qualified_table_names** を無効にする場合、接続 **start_query** 設定で Postgres の適切な **search_path** も設定する必要があります。例

```
[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` で設定された 2 つ以上のスキーマに同じ名前のテーブルが含まれる場合、最初に見つかったもののみが比較の対象となります。

Table Filter セクションには、`publications`、`replication_sets`、`schemas`、および `tables` フィルターを組み合わせることができます。LiveCompare は、指定したすべてのフィルタの交差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
```

テーブルフィルタは、最初にデータベースに適用されてテーブルリストを構築します。テーブルが最初のデータベースに存在し、フィルタ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、次のようなものがログに追加され、その特定のテーブルの比較がスキップされます。

```
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するが、最初のデータベースに存在しない場合、テーブルフィルタを適用しなかった場合でも、比較では考慮されません。

テーブルの列名が完全に同じでなく `column_intersection` が有効になっていない限り、同じ順序である場合、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追加されます。

現在、LiveCompare は、両方のテーブルでデータ型または制約が同じであるかどうかをチェックしません。

重要: `conflicts` モードはテーブルフィルターを利用しません。

9.6 行フィルタ

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

このセクションの下に、テーブルのリストを 1 行に 1 テーブルずつ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されている必要があります。例

[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 id = 10 句を満たす行でのみ機能します。テーブル public.table2 の場合、logdate >= '2000-01-01' を満たす行のみが比較の対象となります。

一般設定 schema_qualified_table_names を無効にする場合、接続 start_query 設定で Postgres の適切な search_path も設定する必要があります。例

[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
```

あらゆる種類の SQL 条件 WHERE 句に入力するのと同じものは、テーブル行フィルタと同じ行で受け入れられます。たとえば、大規模なテーブルがあり、特定の数の ID のみを比較する場合は、すべての ID を使用して一時テーブルを作成できます。次に、次のように IN 句を使用して JOIN をエミュレートできます。

[Row Filter]

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

行フィルタが誤って書き込まれている場合、LiveCompare はフィルタ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、どういうわけか Table Filter によってフィルタリングされている場合、このテーブルの行フィルタはサイレントに無視されます。

重要: conflicts モードは行フィルターを利用しません。

9.6.1 行フィルタで現在のタイムスタンプを使用する

Row Filter は、`data_fetch_mode` に応じ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、クエリが実行されるテーブル比較の先頭にのみ Row Filter が適用されます。これは、サーバーサイドのカーソルを使用してデータを取得すると、データが比較の開始時の状態のスナップショットとして表示され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、準備されたクエリーに Row Filter が含まれ、取得されるすべてのデータバッファで実行されます。これは、クエリが Row Filter で `now()`、`CURRENT_TIMESTAMP`、または `SYSDATE` EDB Postgres Advanced Server でを使用する場合、準備されたステートメントが実行されると、Postgres が現在のタイムスタンプを再評価することを意味します。

したがって、Row Filter で `now()`、`CURRENT_TIMESTAMP`、または `SYSDATE` を使用しているとします。たとえば

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

この場合、サーバーサイド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、`data_fetch_mode` 設定は無視され、クエリーは最初に実行されます。次に、データはクライアントサイドのカーソルを介して取得されます。このアプローチにより、データは比較の開始時の状態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サイドのカーソルですが、動作は Postgres でサーバーサイドのカーソルを使用するのと似ています。

9.7 列フィルタ

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。

このセクションの下に、テーブルのリストを 1 行に 1 テーブルずつ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されている必要があります。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 は列 `column2`、`column4`、および `column5` でのみ機能し、`column1` および `column3` をフィルタリングします。テーブル `public.table2` の場合、列 `column2`、`column3`、および `column4` のみが比較で考慮され、`column1` および `column5` がフィルタリングされます。

一般設定 `schema_qualified_table_names` を無効にする場合、接続 `start_query` 設定で Postgres の適切な `search_path` も設定する必要があります。例

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

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

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

存在しない列名が列フィルタで指定されている場合、つまり、列が指定されたテーブルに存在しない場合、LiveCompare は欠落している列に関するメッセージをログに記録し、それらを見捨てます。有効なものがある場合のみを使用します。

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** によってフィルタリングされている場合、このテーブルの列フィルタはサイレントに見捨てられます。

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、比較で見捨てられません。LiveCompare はそれをログに記録し、このような列のフィルターを見捨てます。

重要: *conflicts* モードは列フィルターを利用しません。

9.8 比較キー

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

例

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

この例では、テーブル `public.table1` の場合、比較キーは列 `col_a` および `col_b` です。テーブル `public.table2` の場合、列 `c1` および `c2` は比較キーとして考慮されます。

列フィルタ で説明している欠落している列、またはフィルタリングまたは欠落しているテーブルに関する同じ動作も、比較キーに適用されます。同様に、

Comparison Key セクションは `conflicts` モードでは無視されます。

9.9 競合フィルター

このセクションでは、PGD 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ーを指定できます。任意の SQL 条件式を構築し、式でこれら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` 競合のタイプ。
- `conflict_resolution` 適用された解像度。
- `nspname` 関連するリレーションのスキーマ名。
- `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。例

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

この構成の部分を `.ini` ファイルに追加すると、LiveCompare は、各 PGD ノードで競合を照会しているときに、タイプ `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` モード専用です。

第10章 Licenses

10.1 TQDM

tqdm は共同作業の製品です。特に記載のない限り、すべての作者コミットログを参照してください、各作品の著作権を保持し、MIT ライセンスの下で作品をリリースします。以下のテキスト。

例外または重要な著者を新しい順に以下にリストします。

- 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. に代わって
- ファイル tqdm/_tqdm.py setup.py README.rst MANIFEST.in .gitignore MIT 2013 (c) Noam Yorav-Raphael、原作者。

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

10.1.1 Mozilla Public LicenseMPL v. 2.0 - 証拠 A

このソースコードフォームには、Mozilla Public License v. 2.0 の条件が適用されます。MPL のコピーがこのファイルとともに配布されていない場合は、<https://mozilla.org/MPL/2.0/> で入手できます。

10.1.2 MIT ライセンス MIT

Copyright (c) 2013 noamraph

本ソフトウェアおよび関連ドキュメントファイル「ソフトウェア」のコピーを取得した人に、使用、コピー、変更、マージの権利を含むがこれに限定されない、制限なしでソフトウェアを取り扱うことを無料で許可します。本ソフトウェアのコピーを出版、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そうすることを許可します。

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。

ソフトウェアは、明示または黙示にかかわらず、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ない、いかなる種類の保証もなく、「現状のまま」提供されます。いかなる場合も、作者または著作権者は、契約行為、不法行為またはその他の行為であるかどうかにかかわらず、ソフトウェアまたはソフトウェアまたはその他の取引に関連して、いかなる請求、損害、またはその他の責任についても責任を負いません。ソフトウェア。

10.2 cx_Oracle

CX_ORACLE のライセンス契約

Copyright © 2016、2020、Oracle および/またはその関連会社。すべての著作権は留保されます。

著作権© 2007-2015、アンソニー・トゥイニング。すべての著作権は留保されます。

著作権© 2001-2007、Computronix (Canada) Ltd.、カナダ、アルバータ州エドモントン。すべての著作権は留保されます。

変更の有無にかかわらず、ソースおよびバイナリ形式での再配布と使用は、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、この条件リスト、およびそれに続く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、この条件リスト、および配布と共に提供されるドキュメントおよび/またはその他のマテリアルにある次の免責事項を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、特定の事前の書面による許可なしに、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ために使用してはなりません。免責事項このソフトウェアは、著作権者および貢献者によって提供されています**現状のまま**、および商品性および特定の目的への適合性の暗黙の保証を含むがこれに限定されない明示または黙示の保証は免責されます。いかなる場合も、リジェントまたは貢献者は、直接、間接、付随的、特別、例示的、または結果的損害代替品またはサービスの調達、使用、データ、または利益の損失を含みますが、これらに限定されません、または業務中断) たとえ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、本ソフトウェアの使用から何らかの形で発生した契約、厳密責任、または不法行為過失またはその他を含みます。

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

© Copyright 2016、2020、Oracle および/またはその関連会社。すべての著作権は留保されます。部分の著作権 © 2007-2015、Anthony Tuininga。すべての著作権は留保されます。部分の著作権 © 2001-2007、Computronix (Canada) Ltd.、カナダ、アルバータ州エドモントン。すべての著作権はリビジョン 10e5c258 に留保されます。

10.2.1 Apache ライセンス

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

使用、複製、および配布の利用規約

1. 定義。

「ライセンス」とは、このドキュメントのセクション 1 から 9 で定義される使用、複製、および配布の利用規約を意味します。

「ライセンサー」とは、ライセンスを付与している著作権所有者、または著作権所有者によって承認されたエンティティを意味します。

「法的エンティティ」とは、代理エンティティと、そのエンティティを制御する、制御される、または共通の制御下にあるその他のすべてのエンティティの組合を意味します。この定義では、「支配」とは、(i) 契

約かその他によるかにかかわらず、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直接または間接的な権限、または (ii) 50% または 50% の所有権を意味します。発行済み株式のより多く、または (iii) そのようなエンティティの受益所有権。

「あなた」または「あなたの」とは、このライセンスによって付与された権限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、構成ファイルを含むがこれらに限定されない、変更を行うための優先フォームを意味します。

「オブジェクト」フォームとは、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を含むがこれらに限定されない、ソースフォームの機械的変換または翻訳の結果として生じるフォームを意味します。

「作品」とは、ソースまたはオブジェクトの形式にかかわらず、著作物に含まれるまたは添付される著作権表示で示される、ライセンスに基づいて利用可能な作者の作品を意味します。以下の付録に例を提供します。

「派生作品」とは、ソース形式であれオブジェクト形式であれ、作品に基づいてまたは派生し、編集修正、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。このライセンスの目的では、派生作品には、作品およびその派生作品のインターフェイスに分離可能なままであるか、単にリンクまたは名前でバインドされている作品は含まれないものとします。

「コントリビューション」とは、作品のオリジナルバージョンと、その作品またはその派生作品への変更または追加を含む、著作権所有者または個人またはによって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著作物を意味します。著作権所有者に代わって送信する権限を有する法人。この定義では、「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、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。は、作品を議論および改善する目的で、ライセンサーによって、またはそれに代わって管理されますが、目立つようにマークされている、または著作権所有者が「非貢献」として書面で指定しているコミュニケーションは除きます。

「コントリビューター」とは、ライセンサーと、そのコントリビューションがライセンサーによって受け取られ、その後作品内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。本ライセンスの利用規約に従って、各コントリビューターは、ここに、複製、派生作品の準備、公に表示、公に実行するための永久、世界的、非独占、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。サブライセンスを取得し、作品およびその派生作品をソースまたはオブジェクトの形式で配布します。
3. 特許ライセンスの付与。このライセンスの利用規約に従って、各コントリビューターは、永久、世界的、非独占、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きます特許ライセンスをあなたに付与します。作品の使用、販売の申し出、販売、輸入、およびその他の方法での譲渡。ここで、そのようなライセンスは、そのコントリビューターがライセンス可能な特許請求にのみ適用され、コントリビューションを単独でまたはコントリビュートとの組み合わせによって必然的に侵害されていますかかる投稿が提出された作品。あなたが、作品または作品内に組み込まれている貢献が直接または寄与的な特許侵害を構成すると主張して、エンティティに対して特許訴訟訴訟の相互請求または反訴を含む場合、本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、かかる訴訟が提起された日をもって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に、作品またはその派生作品のコピーを、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領者に、このライセンスのコピーを提供しなければなりません。そして
- (b) 変更したファイルには、ファイルを変更したことを記載した目立つ通知を掲載させなければなりません。そして
- (c) 派生作品のどの部分にも関係しない通知を除き、作品のソースフォームからのすべての著作権、特許、商標、および帰属通知を、配布する派生作品のソースフォームで保持する必要があります作品;そして
- (d) 作品がその配布の一部として「NOTICE」テキストファイルを含む場合、配布する派生作品には、かかる 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。派生作品の一部、次の場所の少なくとも 1 つ。派生作品の一部として配布される通知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、派生作品によって生成された表示内で、このようなサードパーティの通知が通常表示される場合と場所。NOTICE ファイルの内容は情報提供のみを目的としており、ライセンスを変更するものではありません。あなたは、配布する派生作品内に、作品の通知テキストと並んで、または補遺として、独自の帰属通知を追加できます。ただし、このような追加の帰属通知は、ライセンスを変更するものとして解釈されないことを条件とします。

あなたは、あなたの変更に独自の著作権声明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項および条件を提供することができます。それ以外の場合、作品はこのライセンスに記載されている条件に準拠します。

- 5. 投稿の提出。あなたが明示的に別の記載をしない限り、作品に含めるためにあなたがライセンサーに意図的に提出した投稿は、追加の契約条件なしで、このライセンスの契約条件の下にあるものとします。上記にかかわらず、本契約のいかなる規定も、かかるコントリビューションに関してライセンサーと締結した個別のライセンス契約の条件に優先したり、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、通知ファイルのコンテンツを複製する際に合理的かつ慣例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. 保証の免責事項。適用法で要求されるか、書面で合意しない限り、ライセンサーは、明示または暗黙にかかわらず、いかなる種類の保証または条件なしで、作品を提供し、各コントリビューターはそのコントリビューションを提供します。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。あなたは作品を使用または再配布することの適切性を判断することに単独の責任を負い、このライセンスに基づく権限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、法理論に基づいて、不法行為過失を含むか、契約その他の場合にかかわらず、適用される法律故意および重大な過失行為などで要求される場合または書面で同意される場合を除き、コントリビューターは以下についてあなたに対して責任を負わないものとします。このライセンスの結果、または作品の使用または使用不能から発生したキャラクターの直接的、間接的、特別、付随的、または結果的損害を含む損害賠償善意の喪失、作品の停止に対する損害賠償を含みますが、これらに限定されません、コンピューターの障害または誤動作、またはその他の商業的損害または損失)、かかる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間、あなたは、このライセンスと一致するサポート、保証、補償、またはその他の責任義務および/または権利の受諾を提供し、料金を請求することを選択できます。ただし、そのような義務を受け入れる際に、あなたは他の貢献者に代わってではなく、自分自身に代わって自分の単独の責任でのみ行動することができます。

または、お客様がかかる保証または追加の責任を受け入れたことを理由に、かかる寄稿者に対して主張された請求。

利用規約の終わり

10.3 Psycpg2

psycpg2 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 Lesser General Public License の条件、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それ以降のバージョンに基づいて、再配布および/または変更ができます。

psycpg2 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暗黙の保証さえなし。詳細については、GNU Lesser General Public License を参照してください。

さらに、特別な例外として、著作権者は、このプログラムを OpenSSL ライブラリまたは OpenSSL と同じライセンスを使用する OpenSSL の変更バージョンとリンクし、2 つを含むリンクされた組み合わせを配布することを許可します。

OpenSSL 以外に使用されるすべてのコードは、あらゆる点で GNU Lesser General Public License に従う必要があります。この例外で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、そうする義務はありません。これをしたくない場合は、バージョンからこの例外ステートメントを削除します。プログラム内のすべてのソースファイルからこの Exception ステートメントを削除する場合、ここでも削除します。

psycpg2 doc/ディレクトリを参照してください。そうでない場合は、<https://www.gnu.org/licenses/> を参照してください。

10.3.1 代替ライセンス

次の BSD のようなライセンスは、パターン `psycpg/adapter *.{h,c}` および `psycpg/microprotocol*. {h,c}` に従っているファイルに適用されます。

次の制限を条件として、商用アプリケーションを含むあらゆる目的でこのソフトウェアを使用し、自由に変更および再配布することが誰にでも付与されます。

1. このソフトウェアの出所を偽って伝えてはなりません。オリジナルのソフトウェアを自分が書い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、製品ドキュメントの謝辞はいただければ幸いですが、必須ではありません。
2. 変更されたソースバージョンは、そのように明確にマークされ、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してはなりません。

10.4 集計する

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

本ソフトウェアおよび関連ドキュメントファイル「ソフトウェア」のコピーを取得した人に、使用、コピー、変更、マージの権利を含むがこれに限定されない、制限なしでソフトウェアを取り扱うことを無料で許可します。、本ソフトウェアのコピーを出版、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そうすることを許可します。

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します。

ソフトウェアは、明示または黙示にかかわらず、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ない、いかなる種類の保証もなく、「現状のまま」提供されます。いかなる場合も、作者または著作権者は、契約行為、不法行為またはその他の行為であるかどうかにかかわらず、ソフトウェアまたはソフトウェアまたはその他の取引に関連して、いかなる請求、損害、またはその他の責任についても責任を負いません。ソフトウェア。© 2020 GitHub, Inc.

10.5 OmniDB

MIT ライセンス

ポーションの著作権 c 2015-2019、The OmniDB Team ポーションの著作権 c 2017-2019、2ndQuadrant Limited

本ソフトウェアおよび関連ドキュメントファイル「ソフトウェア」のコピーを取得した人に、使用、コピー、変更、マージの権利を含むがこれに限定されない、制限なしでソフトウェアを取り扱うことを無料で許可します。、本ソフトウェアのコピーを出版、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そうすることを許可します。

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します。

ソフトウェアは、明示または黙示にかかわらず、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ない、いかなる種類の保証もなく、「現状のまま」提供されます。いかなる場合も、作者または著作権者は、契約行為、不法行為またはその他の行為であるかどうかにかかわらず、ソフトウェアまたはソフトウェアまたはその他の取引に関連して、いかなる請求、損害、またはその他の責任についても責任を負いません。ソフトウェア。