



EDB

LiveCompare

リリース 2.1.0

EDB

2022年04月01日

目次

第 1 章	比較パフォーマンス	3
第 2 章	ユーザーのセキュリティに関する考慮事項	5
2.1	Release Notes	5
2.1.1	2.1.0 (2022-03-31)	5
2.1.2	2.0.0 (2022-02-15)	6
2.1.3	1.18.1 (2021-12-14)	8
2.1.4	1.18.0 (2021-12-01)	8
2.1.5	1.17.0 (2021-10-01)	9
2.1.6	1.16.0 (2021-08-04)	9
2.1.7	1.15.0 (2021-06-06)	10
2.1.8	1.14.0 (2021-05-14)	10
2.1.9	1.13.1 (2021-04-14)	11
2.1.10	1.13.0 (2021-02-25)	11
2.1.11	1.12.0 (2021-02-11)	12
2.1.12	1.11.0 (2021-01-19)	13
2.1.13	1.10.1 (2020-10-29)	14
2.1.14	1.10.0 (2020-10-02)	15
2.1.15	1.9.1 (2020-08-13)	15
2.1.16	1.9.0 (2020-08-06)	15
2.1.17	1.8.0 (2020-06-15)	17
2.1.18	1.7.0 (2020-05-18)	18
2.1.19	重大な変更	18
2.1.20	1.6.0 (2020-04-03)	20
2.1.21	1.5.0 (2020-03-13)	20
2.1.22	1.4.0 (2020-02-13)	21
2.1.23	1.3.0 (2020-01-30)	21
2.1.24	1.2.0 (2020-01-03)	22
2.1.25	1.1.0 (2019-12-18)	23
2.1.26	1.0.0 (2019-12-03)	23
2.1.27	0.12.0 (2019-11-01)	24
2.1.28	0.11.0 (2019-10-10)	24
2.1.29	0.10 (2019-09-24)	25
2.1.30	0.9 (2019-08-30)	27
2.1.31	0.8 (2019-08-07)	28
2.1.32	0.7 (2019-08-06)	28
2.1.33	0.6 (2019-06-17)	28
2.1.34	0.5 (2019-06-17)	28
2.1.35	0.4 (2019-06-12)	28
2.1.36	0.3 (2019-05-29)	28

2.1.37	0.2 (2019-05-21)	29
2.1.38	0.1 (2019-05-17)	29
2.2	Requirements	29
2.2.1	TPAexec との LiveCompare	30
2.3	Supported Technologies	31
2.3.1	PgBouncer のサポート	32
2.4	Command-line Usage	32
2.4.1	比較モード	32
2.4.2	再確認モード	34
2.4.3	競合モード	34
2.5	Advanced Usage	35
2.5.1	比較中絶	36
2.5.2	比較キー	37
2.5.3	修正する違い	37
2.5.4	差分ログ	38
2.6	BDR Support	38
2.6.1	BDR 監視ノード	40
2.6.2	BDR クラスターの違い	41
2.6.3	BDR の競合	43
2.6.4	ミキシング技術	47
2.7	Oracle Support	47
2.7.1	要件	50
2.7.2	違い	51
2.7.3	BLOB および CLOB データ型	51
2.7.4	互換性のない照合	52
2.7.5	共通ハッシュ	52
2.8	Settings	53
2.8.1	一般設定	53
2.8.2	初期接続	60
2.8.3	出力接続	60
2.8.4	データ接続	61
2.8.5	テーブルフィルター	62
2.8.6	行フィルター	64
2.8.7	列フィルター	65
2.8.8	比較キー	66
2.8.9	競合フィルター	66
2.9	Licenses	67
2.9.1	TQDM	67
2.9.2	MIT ライセンス (MIT)	67
2.9.3	cx_Oracle	68
2.9.4	Psycopg2	71
2.9.5	表	72
2.9.6	OmniDB	72

© Copyright EnterpriseDB UK Limited 2019-2021-無断複写・禁無断転載

#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、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、ユーザはオプションで DML を適用し、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、マルチプルのテーブル（マルチプルのワーカースプロセス）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スタートされ、テーブルごとの比較がスタートされます。スタンドアローンモードでは、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。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。

各テーブル比較オペレーションは「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 はテーブルをマルチプルの比較ラウンドに分割し、同時に他のワーカーによって実行されている他のテーブルと並行して実行されます。

スタンドアローンモードでは、テーブルの最初の比較ラウンドは、テーブルの最初（最も古い既存の PK）からテーブルの最後（最も新しい既存の PK）まで始まります。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、各テーブルから最小および最大 PK を取得オーダーに PK 列をソートします。ソートできない各 PK 列について、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、オラクルでは `to_char` を使用して実現しています。

比較アルゴリズムを実行するとき、各ワーカーは $N + 1$ 個のデータベース接続を必要とし、 N は比較されるデータベースの数です。追加の必要な接続は、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。

比較アルゴリズムで見つかった違いは、後で都合の良いときにユーザが手動で再確認できます。これは、レプリケーションの整合性チェックを許可するために行うことをお勧めします。差分の再チェック時に、レプリケーションがその特定の行に追いつき、差分がもう存在しないため、差分は削除され、そうでない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 スクリプトを生成して、ユーザがそれを確認し、差異を 1 つずつ修正するか、DML スクリプト全体を適用して永続的な差異をすべて修正できるようにします。

LiveCompare は、行レベルでロジカルデータの整合性を保証に潜在的に使用できます。例、次のシナリオの場合：

- データベース技術の移行（オラクル 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後、例、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。
- フェイルオーバー後に予期しないスプリットブレインシチュエーションが発生した場合。古い場合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みました。LiveCompare を使用して、古いデータに存在するデータを正確に知ることができます。プライマリであり、新しい

プライマリには存在しません。必要に応じて、DBA はこれらのデータを新しいプライマリに適用するために LiveCompare が生成する DML スクリプト。

- 論理レプリケーション。3 種類のロジカルレプリケーションテクノロジーはサポート：Postgres ネイティブロジカルレプリケーション、pglogical および BDR。

第1章 比較パフォーマンス

LiveCompare は、稼働システムでの使用に最適化されており、後述のチューニング用のさまざまなパラメータがあります。比較ラウンドは読み取り専用のワークロードです。使用例では、4つの接続と4つのワーカーで9分17秒の6つのテーブルの43,109,165行を比較し、1秒あたり約77k行、または4時間以内に10億行のパフォーマンスを比較しました。

上記のユースケースは、一般的なユースケースと考えることができます。低負荷、テスト、移行、およびその他の特定のシナリオでは、`data_fetch_mode` 設定を変更してサーバサイドカーソルを使用することで速度を改善できる場合があります。私たちの実験では、サーバサイドカーソルの種類ごとに、小さなテーブルまたはラージテーブルが関係するユースケースでパフォーマンスが向上します。

第2章 ユーザーのセキュリティに関する考慮事項

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

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

BDR で DML スクリプトを適用するには、すべての分岐接続（場合によってはすべてのデータ接続）で、`bdr.xact_replication` を無効にオーダーに `bdr_superuser` が付与されているユーザーが必要です。

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` というレプリケーションオリジンに関連付けます。LiveCompare は、過去のトランザクション日時の 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 BDR の競合がある場合、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、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できないオーダーに、適用スクリプトのトランザクションはロールをテーブルの所有者に変更します。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スイッチことができるニーズがあります。

2.1 Release Notes

2.1.1 2.1.0 (2022-03-31)

新機能

- Postgres- BDR 4 (LIV-131) のサポート。
- 新しい設定 `min_time_between_heart_beats`。LiveCompare に、heart beat ごとに比較の進行状況を記録します。デフォルトでは、INFO ログログレベル (LIV-128) を使用して 30 秒に設定されます。
- 新しい設定 `comparison_cost_limit` および `comparison_cost_delay`。0 より大きい、`comparison_cost_limit` の行数 (LIV-16) を処理した後、各ワーカーに `comparison_cost_delay` 秒 (例 0.5) の仮眠を取るよう指示します。

その他の変更

- `parallel_chunk_rows` のデフォルト値は `0` に設定されています。これにより、デフォルトでテーブル分割が無効になります。最近の調査により、一般的なユースケースでパフォーマンスが低下することが判明したためです。詳細については、設定の文書 (LIV-130) を参照してください。
- `CanAdvanceCursors` メソッド (LIV-129) から処理された行の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

- `ORA-01830` (LIV-127) で失敗する `timestamp(6)` データタイプのオラクルと Postgres の比較の問題を修正しました。

2.1.2 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** という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、プライマリキーまたは一意性インデックス (LIV-5) のないテーブルに役立ちます。
- テーブルにユーザー定義の比較キーまたはプライマリキーがない場合、LiveCompare はテーブルの一意性インデックスを使用しようとします。テーブルにはマルチプルの一意性インデックスを含めることができるため、LiveCompare は、列を `NULL` にできないインデックスを使用することを好みます。可能でない場合、LiveCompare は列の少ない最初の一意インデックスを使用しようとします。一意性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、一意性インデックス (LIV-39) からの `NULL` 入力可能列を無視しようとしないことにノートしてください。
- 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 はすべての列を考慮する際の誤検知を避けるために、`null` 許容列を無視するようになりました。この動作を無効にするには、`ignore_nullable = false` (LIV-96) を設定します。
- 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確認します。列セットを使用するとき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。この動作を無効にするには、`check_uniqueness_enforcement = false` (LIV-119) を設定します。

パフォーマンスの改善

オラクル 12c 以降では、LiveCompare はすでに共通のハッシュを使用して `comparison_algorithm = block_hash` と `comparison_algorithm = row_hash` を許可しており、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、オラクル 12c で導入された `standard_hash()` ファンクションを使用してオラクル側で行われます。LiveCompare では、ユーザーが `sys.dbms_crypto` オラクルシステムパッケージ (LIV-79) に対する `EXECUTE` 特権を持っている場合、同等のファンクション `sys.dbms_crypto.hash()` を使用して、オラクル 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

オラクル 21c (LIV-47) のサポートが追加されました。- レプリケーション対応 (BDR、pglogical または ネイティブロジカルレプリケーション) データベース (LIV-42) に LiveCompare スキーマを追加できるようになりました。- LiveCompare は、PostgreSQL 14 (LIV-73) の新しい `pg_read_all_data` ロールを使用できます。- 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 (LIV-98) を生成する際のエラーを回避するために、Unicode シーケンス `\u0000` を適切にクォートしてください。
- テーブルがマルチプルのラウンドパーツに分割されたときに、合計行数が誤って表示される問題を修正しました (LIV-14)。
- Oracle の空の BLOB と Postgres の空のオラクルを比較すると、誤検知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例外が発生し、比較セッション全体が中止される問題を修正しました。現在、LiveCompare は再接続して、可能であれば比較を続行できます (LIV-84)。
- 分岐がある場合の `recheck` モードでの未処理の例外を修正しました (LIV-107)。
- テーブルが接続上に存在せず、`logical_replication_mode` が無効になっている場合にテーブル比較が中止されなかった問題を修正しました (LIV-108)。
- `timestamp` データタイプのフィールドが常にオラクルと 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。
- 無視された列がまだ一般的なハッシュで考慮されている問題を修正しました。

廃止のお知らせ

オラクル 10g (LIV-47) のサポートを終了しました。

2.1.3 1.18.1 (2021-12-14)

バグ修正

- 非同期接続 (LIV-95) で内部のラウンドパート列名前の設定が考慮されていなかった問題を修正しました。

2.1.4 1.18.0 (2021-12-01)

パフォーマンスの改善

- Oracle 用の新しいオラクル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。これは、オラクルが、最後に処理された複数列 PK 値を超えて行をフェッチする必要があった行値式を含む不等式演算子を許可しないため、より効率的です。これにより、オラクルと Postgres の比較のパフォーマンスが大幅に向上します。PostgreSQL は行値式を含む不等式演算子を許可するため、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、肥大化と xmin ホライズンの抑制のインパクトを与えません。したがって、Postgres と Postgres のパフォーマンスは、この実装 (LIV-83、RT75644、RT75675、RT75332) の影響を受けません。
- `comparison_algorithm = block_hash` を使用する場合にバッファをリセットする必要がなくなりました。ハッシュが一致しないブロックが見つかったと、LiveCompare は `comparison_algorithm = row_hash` に正しく切り替えられ、一致しないブロック内のすべての行 (既にフェッチ済み) が個別に比較され、`comparison_algorithm = block_hash` に戻って次のバッファをフェッチします。これにより、オラクルと 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` に設定されている場合 (現在オラクル 12 以降が必要)、LiveCompare はオラクルと Postgres の比較でもラージテーブルを分割できるようになりました。以前は、ハッシュはデータベース側で行われるため、Postgres と Postgres でのみ可能でした。オラクルと Postgres は共通のハッシュファンクションを共有していないため、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、MD5 に基づく共通の行ハッシュを整数ハッシュに変換しました。オラクルでは、パフォーマンス上の理由により、データベース側では実行できなかったため、Python 側でこのハッシュを計算します。オラクルと 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、オラクルと Postgres の場合、`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。例、1 億以上に設定すると、アルゴリズムは非常にラージテーブル (LIV-87) のみを分割します。

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。 - 分岐の検査に役立つオラクル `rowid` の収集 (LIV-90、RT75644、RT75675)。 - 相違の検査に役立つ LiveRound 比較アルゴリズムのロギングの改善 (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` を使用している場合、ロギングキューがいっぱいであるため、LiveCompare が未処理の例外で終了す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。問題は、ベースの Python 実装がロギングレコードを非同期的にキューに登録したために発生しました。解決策は、Python 実装をオーバーライドしてロギン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) の場合にパフォーマンスにインパクト可能性があります。

2.1.5 1.17.0 (2021-10-01)

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 (LIV-76) を必要とします。
- 文書のマイナーな修正 (LIV-26、LIV-68、LIV-80)。

2.1.6 1.16.0 (2021-08-04)

パフォーマンスの改善

オラクルと Postgres の比較セッションでは、MD5 に基づく一般的な行ハッシュを使用できるため、一般的なユースケースのパフォーマンスが向上します。以前は、オラクルと Postgres の比較は `comparison_algorithm = full_row` に制限されていましたが、`comparison_algorithm = row_hash` または `comparison_algorithm = block_hash` を使用できるようになりました。デフォルトでは、`comparison_algorithm = block_hash` に設定されています。オラクルの制限により、Common Hash、BLOB および CLOB フィールドの使用は 2000 文字に制限されており、行全体が 4000 文字を超えることはできません。Common Hash の要件を満たしていないテーブルは、`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、文書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。

2.1.7 1.15.0 (2021-06-06)

改善

- `--conflicts` 実行モードの競合フィルタリングを導入 (LIV-40)
- 照合順序が指定されていない場合 (LIV-44)、LiveCompare は混在テクノロジー比較 (オラクルと PostgreSQL など) で `force_collate = C` を想定します。
- すでに追加された EPAS 13 のサポートに加えて、EPAS 11 および 12 (LIV-48) のサポートが追加されました。

バグ修正

- Oracle と PostgreSQL の比較でオラクルがスタックしなくなりました (RT71243 / LIV-43)
- `--conflicts` (LIV-50) で作成された比較セッションで `--recheck` を実行できなかったバグを修正しました。

2.1.8 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、競合のログに記録された行 (`key_tuple`、`local_tuple`、フィルタリング、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自動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チェックされます。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、実行可能であり、クラスターの一貫性を保証に頻繁に実行することをお勧めします (LIV-28)。

改善

- LiveCompare は、特定のテーブルが存在しない、またはレプリケーションされたデータを受信しない BDR ノードを無視できるようになりました。これまでは、すべての接続で比較が実行されていましたが、比較コンセンサスにより、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。テーブルが存在するノード間、およびノードがサブスクライブされているレプリケーションセットにテーブルが関連付けられているノード間で、コンセンサスマジョリティが引き続き達成されます。これにより、BDR 監視ノード (LIV-29) を適切に無視するなど、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。
- `--help` のメッセージが改善され、すべての実行モードの簡単な説明が表示されました (LIV-28)。
- 列が NULL 入力可能かどうかに関する情報は、テーブルメタデータ (LIV-28) でも収集されるようになりました。

- 列がプライマリキー (LIV-4) のパートである場合、ソートできないデータ型が `string` に変換されることをドキュメントで明確にしました。
- `oracle_ignore_unsortable` がオラクルの PK のパートではないソート不能な列を無視することをドキュメントで明確にしました (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` を有効にできること (デフォルトでは) 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 (LIV-26) であることをドキュメントで明確化。

バグ修正

- LiveCompare は現在、型をスキーマ修飾名としてハンドリングしています。この修正 (LIV-35) の前に、`search_path` からの型は LiveCompare で検出できませんでした。

廃止

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 スクリプトに含まれています。これは文書 (LIV-1、LIV-31) で明確にされています。

2.1.9 1.13.1 (2021-04-14)

バグ修正

オラクルデータ型 `RAW` および `ROWID` のサポートの問題を修正 (RT69656)。オラクルからのバイナリオブジェクトフィールドが `NULL` であった場合の未処理の例外を修正しました (RT69656)。オラクル x Postgres の比較サポートに関する問題を修正しました。ドル引用符で囲まれた文字列が含まれます (RT69656)。- 特定の PostgreSQL 予約キーワードとして名前付けされた列が引用され、比較で考慮されなかったコーナケースを修正しました (RT69656)。

2.1.10 1.13.0 (2021-02-25)

パフォーマンスの改善

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。次の値のいずれかを想定できます。- `prepared_statements`: 準備されたステートメントを使用してデータをフェッチする、このバージョンと以前のバージョンのデフォルトの動作。データベース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を発揮します。稼動データベースにのみ推奨されます。- `server_side_cursors_with_hold`: サーバサイドカーソル `WITH HOLD` を使用してデータをフェッチします。これにより、LiveCompare に送信される前にデータの各チャンクが一時ファイルに実体化されるため、`xmin` を抑制してデータベースのメモリと一時ファイルの使用量を増やすことができます。その結果、マ

ルチブルの小さなテーブルでより高速になる可能性があります、テストおよび移行シナリオで推奨されます。- `server_side_cursors_without_hold`: サーバサイドカーソル `WITHOUT HOLD` を使用してデータをフェッチします。これにより、`xmin` を抑制できますが、データベース上のメモリおよび一時ファイルの使用量は `WITH HOLD` よりも低くなります。ラージテーブルの場合は高速になる可能性があります、低ロードのデータベースにはお勧めです。

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、比較セッション全体をアボートするよう LiveCompare に指示します。デフォルトでは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` により、設定 `setup_max_attempts` および `setup_min_interval_between_attempts` は廃止されました。

2.1.11 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、適用スクリプトの開始時に `session_replication_role` が `replica` に設定されます (RM20247、RT69742)。
- 新しい設定 `split_updates` を実装しました。有効にすると、`UPDATE` ステートメントが対応する `DELETE` ステートメントと `INSERT` ステートメントに分割されます (RM20355、RT69742)。
- 新しい接続設定 `fetch_fk_metadata`。外部キーに関するメタデータを特定の接続で取得する必要があるかどうかを示します。デフォルトで有効。オラクル接続の `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`。これは、浮動小数点値を比較するときに LiveCompare に指定された 10 進数数で丸めるように指示する場合。デフォルトでは無効になっています (RT69381)。

削除された機能

- `--fix` 実行モードが削除されました。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。

改善

- LiveCompare によって生成された適用スクリプトには、最初に `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します。
- すべての接続で比較されているテクノロジーが PostgreSQL である場合、`comparison_algorithm = full_row` は `parallel_chunk_rows` を尊重し、テーブルをマルチプルのワーカーに分割します。テーブルの分割を無効にするには、`parallel_chunk_rows = 0` を設定します。
- EPAS の比較時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った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により、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が修正されました (RT70032、RM20378)。
- `oracle_ignore_unsortable` または `column_intersection` がすべての列を削除し、LiveCompare が比較を実行しようとする問題を修正しました。LiveCompare はこれらのテーブルの比較を中止し、適切なメッセージを記録するようになりました (RT69656)。
- PK 値を表すために LiveCompare が `custom_dollar_quoting_delimiter` を使用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、順序付けの違いにより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。

2.1.12 1.11.0 (2021-01-19)

改善

- LiveCompare によって生成される DML コマンドのオーダーの新しい設定 `difference_statement_order`。デフォルトでは `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

- カーソルの最小バグ値がすでにデータの終わりに達したカーソルの最大 PK 値よりも大きいためにカーソルがまだ初期化されていない場合、前のカーソルは初期化されるが、そのデータは終了 (RT69742、RM20249)。
- `logical_replication_mode = pglogical` のときに LiveCompare が `Initial Connection` と `node_name` を許可していなかったバグを修正しました (RM19795)。
- `schema_qualified_table_names = off` の場合に強制するのではなく、`schema_qualified_table_names = on` の場合に LiveCompare が `oracle_user_tables_only = on` を強制するバグを修正しました (RT69656)。
- `comparison_algorithm = full_row` の場合、文字列が適切に引用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。

2.1.13 1.10.1 (2020-10-29)

バグ修正

- 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が向上しました。これは、初期メタデータフェッチフェーズで行われます (RM19681、RT69088)。
- 比較セッションを再開できるようになりました。同じメジャーバージョン上にある場合、異なる LiveCompare バージョンで以前に作成された比較セッションに対して実行モード `--fix` および `--recheck` を実行できます。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` を実行すると、セッションは同じメジャーバージョン (RM19686、RT69088) で作成されたため、LiveCompare は `--fix` モードの実行を許可します。
- `Row Filter` を LiveCompare メタデータに保存するときの単一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引用符付け符デリミタ (`custom_dollar_quoting_delimiter` が設定されていない場合) は、実行中にランダム設定値が変更されたために比較セッションの再開が機能しなかった問題に対処する固定ハッシュ文字列になりました (RM19684、RT69091)。
- `custom_dollar_quoting_delimiter` の設定は、`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デリミタ文字として使用します。 (RM19685、RT69091)。

2.1.14 1.10.0 (2020-10-02)

改善

- 新しい一般設定 Postgres コマンドを生成するときに文字列を引用する方法を LiveCompare に伝えます。設定されていない場合（デフォルト）、LiveCompare はランダム文字列を生成してドル引用符付け符のデリミタとして使用します（RM68950、RM19159）。
- LiveCompare は、セッションの最初の実行（セッション作成）と次のセッション（`resume`、`--recheck` または `--fix`）の間で設定も接続属性も変更されていないことを確認します（RM17639）。
- バックグラウンドまたは別のシェルウィンドウで実行している場合、LiveCompare をソフトストップできるようになりました（RM18564）。

バグ修正

- ユーザが `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、LiveCompare が `--recheck` モードで実行しているときに多数の差分を正しく更新していませんでした（RM19158）。
- `--recheck` モードで LiveCompare がマルチプル回実行された場合、`apply_on_the_<node>.sql` ファイル内の複製されたエントリが表示され、サマリーには差分/修正の複製された数が表示されます（RM19176）。
- ラージの相違により、サマリービューで `integer out of range` エラーが発生する（RM18705）。

2.1.15 1.9.1 (2020-08-13)

バグ修正-----

- 現在、特定のプラットフォームに存在しない POSIX 定数をハンドリングしています（RM18197）。
- テーブルパーティションを処理する場合、ユーザ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と比較します。これは修正されました。（RM16912）。

2.1.16 1.9.0 (2020-08-06)

新機能-----

- 設定 `full_comparison_mode = on|off` は廃止され、新しい設定 `comparison_algorithm` に置き換えられました。これにより、新しい `block_hash` アルゴリズム設定が可能になりました-これは、基本テストで 50 % のパフォーマンスを示しました。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、ハッシュに依存しているため、テーブルの分割も無効になります。そのため、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、列ごとに行を比較する

ことにより、完全な比較が実行されます。データ接続が PostgreSQL でない場合、LiveCompare は自動的に `comparison_algorithm = full_row` を設定します。

- **row_hash** (古い `full_comparison_mode = off` と同じ) : ハッシュを使用し行比較を有効にし、テーブル分割を有効にします。テーブルは分割されているため、各ワーカーはテーブルごとに最大 `parallel_chunk_rows` を比較します。PostgreSQL ではデータ行がハッシュされるため、比較は `full_row` より高速です。ただし、特定の行に対してハッシュがマッチしない場合、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、行ごとに比較行ます)。この設定は、すべてのデータ接続が PostgreSQL である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、行行ごとに比較する代わりに、「ブロックハッシュ」、つまり、フェッチされたばかりのデータバッファー内のすべての行のハッシュのハッシュを構築します (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 (このため、この比較アルゴリズムは `row_hash` よりも高速です)。ブロックハッシュがマッチしない場合、LiveCompare は `row_hash` にフォールバックし、バッファー内の行ごとに比較を実行して発散行を見つけます。この設定は、すべてのデータ接続が PostgreSQL である場合にのみ許可されます。これがデフォルト値 (RM14146)。
- LiveCompare は、エラーで終了する前に、各テーブル設定でいくつかの試行を行うことができるようになりました。この動作を構成するために、`setup_max_attempts` (デフォルトは 3) と `setup_min_interval_between_attempts` (デフォルトは 30 秒) の 2 つの新しい設定が追加されました (RM17518)。
- `>` を使用して出力をリダイレクトし、LiveCompare を `ssh` を介して実行している間、LiveCompare 出力が出力ファイルに出力されるようになりました (RM17372)。
- ロギングとデバッグの改善のためにコンセンサスがリファクタリングされました (RM17107)。
- 比較セッションの終わりに、LiveCompare はログレベルに応じて、検出された問題のカウンタを表示するようになりました (RM18196)。
- タイの場合、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、テーブルサマリーで無視される列の最大数を表示するようになりました (RM16581)。
- **Output** または **Initial** (指定されている場合) の場合、LiveCompare は適切なメッセージで中止されます。接続に到達できません。また、到達可能なデータ接続が 2 つより小さい場合は、適切なメッセージで中止されます。2 つ以上が到達可能な場合は、その 1 つだけを比較します (RM14030)。
- Debian および Ubuntu パッケージソフトには、`python3-setuptools` が必要になりました (RM17232)。
- サポートされる最小の 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で明確化されました (RM17232)。
- テーブルパーティションを扱うとき、LiveCompare はマスターテーブルを使用するよりも各パーティションをスキャンすることを好みます。この方法により、行数の推定が向上し、ワーカー間のジョブの分割も改善されます (RM16912)。
- 差分カウントの代わりに POSIX 標準結果コードを使用 (RM18197)。
- TPAexec (RM15312) を使用して LiveCompare をインストールする方法を示すサンプル `config.yml` 例をドキュメントに追加しました。

- `--recheck` および `--fix` モードのロギングの改善 (RM15768)。
- 例外が発生した場合、Python スタックトレースがログに含まれます (RM17158)。
- PgBouncer (RM18019) 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。
- `round_id` および `round_part` フィールドを `vw_running_processes` ビューに追加しました。これは、LiveCompare の実行が cron ジョブ (RM16910) としてスケジュールされている場合の比較の進行状況の確認に役立ちます。

バグ修正-----

- Output Connection で `tuple concurrently updated` が発生するコンセンサス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)。
- 実行中に何らかの種類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)。
- ユーザが `max_parallel_workers & lt;= 0` を設定した場合にハングを回避オーダーに、少なくとも 1 つのワーカーを強制しました (RM17374)。
- 印刷されたテーブルのラージ整数の数値フォーマットを修正しました。LiveCompare は、科学表記法を使用する代わりに、全体の数を表示するようになりました (RM16683)。
- `--fix` モードが誤った差分ステータスを報告していた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メタデータで誤って報告されていた問題を修正しました (RM17663)。
- 異なる真実のソースまたはタイブレーカーで LiveCompare を再実行すると、ダイバージェンスがタイとして誤って報告される問題を修正しました (RM17909)。

2.1.17 1.8.0 (2020-06-15)

新機能-----

- 新しい接続 `difference_sources_of_truth`。どの接続が常にコンセンサスを獲得するかを示すために使用されます。 `consensus_mode` が `source_of_truth` に設定されている必要があります (RM15952)。
- 新しい設定 `work_directory`。セッションフォルダーが作成される場所を示します。cron ジョブとしてスケジュールされた LiveCompare の実行に役立ちます (RM16476)。
- 差分修正 DML スクリプトに書き込まれた DML コマンドの順序は、外部キーをアカウントに入れるようになりました。 `--fix` 実行モードでも同じオーダーが使用されます (RM15766)。
- LiveCompare は、既知の接続 ID のリストに対して `difference_tie_breakers` と `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 を指定すると、エラーがユーザに表示され、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 (RM14480、RM14619、RM14974)。

- 通常のテーブルをプリントする場合、データが `null` の場合は空のセルが印刷されます。代わりに `[null]` を出力するようになりました (RM16686)。

バグ修正-----

- Coverity スキャナで検出されたマイナー問題を修正 (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されるようになり、セッション ID か実行モードかを正しく区別します。
- 表形式のデータを転置フォーマットで印刷する場合、データが `null` の場合、フィールド全体が省略されます。この問題は修正されました (RM16686)。

2.1.18 1.7.0 (2020-05-18)

2.1.19 重大な変更

- LiveCompare 実行モードを実装しました：
- `--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の再チェック/修正の違いはありません。比較中に、検出された各差異は `difference_log` テーブルに保存され、後で再チェックし、オプションで自動修正します。
- `--recheck`：`compare` モードで既に作成されたセッションに対して何度でも実行できます。差異を1つずつ再チェックし、`difference_log` テーブルを更新します。
- `--fix`：`compare` モードですでに作成されたセッションに対して実行できます。差分を1つずつ再チェックし、自動的に修正して、`difference_log` テーブルを更新しようとします。
- 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` であり、再チェックおよび修正モードの場合、ビューは `vw_differences` です。
- 自動修正モードでは、`BDR >= 3.6.18` の場合、LiveCompare は `bdr_local_only_origin` (RM14699) という新しい事前作成されたレプリケーションオリジンを使用します。
- 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。
- `live_mode`、`min_time_between_rounds`、および `max_tail_rounds_before_full_round` の設定を削除しました。
- `show_progressBars` の設定が有効になっている場合 (デフォルトでは)、Python モジュール `tqdm` は < 4.16.0、LiveCompare はアップグレード方法の説明を中止します。

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、LiveCompare は、PK 列が同じである限り、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、すべての接続上のテーブルに存在する列の積部分です (RM14147、RT67064)。

- 新しいセクション **Column Filter** を実装しました。各テーブルで、比較で無視する列のコンマ区切りリストを定義できます。PK のパート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 がいない場合、オラクルのソート不可能なデータ型 (BLOB、CLOB、NCLOB、BFILE) の列を無視するよう LiveCompare に指示します。この設定を有効にする場合は、**column_intersection** (RT67064) も有効にすることをお勧めします。
- 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、LiveCompare にオラクルのログインユーザからのみテーブルメタデータをフェッチするように指示します。これは高速です。また、**Table Filter** -> **tables** は、スキーマ名前なしのテーブル名前前でフィルタリングできます (RT67064)。
- デフォルトで有効になっている新しい一般設定 **schema_qualified_table_names** を実装しました。無効にすると、スキーマ修飾されたテーブル名を使用せずにテーブルを比較できます。オラクル Postgres の比較では、**oracle_user_tables_only** も有効にする必要があります。PostgresPostgres、同じデータベース内であっても異なるスキーマにあるテーブルの比較が可能です。また、**schema_qualified_table_names** が有効な場合、**Table Filter** -> **tables**、**Row Filter**、および **Column Filter** は、スキーマ名前なしのテーブル名前を許可します。(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 **start_query** を使用して **search_path** をクリアし、CVE-2018-1058 (オーダー) から保護します。
- 新しい一般設定 **force_collate** を実装しました。デフォルトでは **off** に設定されています。これは、PostgreSQL で照合順序が強制されないことを意味します。有効な照合順序名前に設定する場合、Postgres に C 以外の照合順序がある場合 (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)、異なる照合順序を持つ Postgres データベースまたはオラクルと Postgres データベースを比較すると便利です。
- 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 は各分岐行の BDR/pglogical オリジン名前をフェッチします (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** にエラーを書き込んで、失敗したワーカーをプールに戻します (RT67064)。
- LiveCompare の一般的な進行状況バーに、比較中のエラーが原因で中断されたテーブルの数が表示されるようになりました。
- 最初に接続を開き、データベース接続を再利用します (RT67042)。
- ユーザが構成ファイルを読み取る権限を持っていない場合のエラーメッセージを明確にしました (RT67064)。
- **Table Filter** および **Row Filter** が、一般設定 **schema_qualified_table_names** が無効になっている限り、スキーマで修飾されたテーブル名が必要であることをドキュメントで明確にし、構成例を提供しました (RM15311、RT67669)。
- 設定ファイルで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止します。
- 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についてドキュメントで明確化。

- `postgres` オペレーティングシステムユーザ向けの `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で明確化。
- 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 は現在のユーザに最新の `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;` 句が含まれるようになりました。

バグ修正-----

- 列名前が Postgres の予約ワードであるために列名前の不一致が誤って評価されたため、オラクルと Postgres のテーブル比較が中止されていた問題を修正しました (RT67064)。オラクル：テキスト列を持つ PK が `ORA-00920: invalid relational operator` エラー (RT67064) を生成する可能性がある問題を修正しました。
- ソート不可能なデータタイプの列が `ORDER BY` 句で適切に処理されていなかったオラクルおよび Postgres の問題を修正しました (RT67064)。
- `difference_allow_null_updates` の設定が誤って解釈されていたため、`UPDATE` が列を `NULL` に設定しなかった DML ジェネレータの問題を修正しました。
- 比較カーソルが適切に進行しなかったコーナーケースを修正しました。
- ユーザ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が修正されました。

2.1.20 1.6.0 (2020-04-03)

PostgreSQL >= 11 の場合、`hashtext` の代わりにビルトイン関数 `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)。オラクル PK の列名のリストに重複した列名があり、エラーと比較が中止されるという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します (RT67042)。- ラウンド状態があまりにも頻繁に保存され、`min_time_between_round_saves` を尊重しない問題を修正しました。パフォーマンスが向上します。- 接続およびクエリーの問題のログギングが改善されました。

2.1.21 1.5.0 (2020-03-13)

- Oracle は `parallel_data_fetch` (RM13714、RT67042、および RT67064) をサポートしていないため、接続の 1 つがオラクルである場合、`parallel_data_fetch` の設定は自動的に無効になりオラクル。
- 実行可能なラウンドの最大数、処理可能な行、および単一の比較セッションで検出できる差異の増加 (`integer` から `bigint` へのデータタイプ) (RM13664 および RT67167)。
- 進行状況バーを ASCII から Unicode に変更しました。
- 5 秒ごとに経過するグローバルプログレスバーの時間を更新します。

- グローバルプログレス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、グローバルプログレスバーに追加の統計（検出された差異の数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る場合でも、比較される予定のテーブルの数によってプロセスの最大数が制限される問題を修正しました。
- テーブルのメタデータを取得する際のログの詳細度を INFO から DEBUG に減らしました。
- マニュアル（Ctrl-C）または自動中断（`stop_after_time`）が発生するログの場合を含む。

2.1.22 1.4.0 (2020-02-13)

- 並列データフェッチを実装して、多元比較のパフォーマンスを改善しました。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` を設定することで無効にできます。
- 行がローカルである、つまり、その `xmin` がレプリケーションオリジンに関連付けられていなかった問題を修正しました（RT66906）。

2.1.23 1.3.0 (2020-01-30)

- 多元比較：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フェッチすることにより行われます。これにより、コンセンサス（定数ベースまたはシンプル多数決の両方がサポートされます）およびタイブレーカー接続のオプションのリストに基づいて、データの不一致を判断できます。新しいビュー `vw_consensus` を使用して、LiveCompare の動作の詳細を確認できます。
- 新しい一般設定 `all_bdr_nodes` が追加されました。有効にすると、ユーザは BDR ノードを指す `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た接続のリストを作成します。`bdr.node_summary` ビューに表示されるように、LiveCompare はノード 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ノートしてください。
- 新しい一般設定 `consensus_mode` が追加されました。これは、データ比較で相違が見つかった場合にどの接続（または BDR ノード）が正しいと見なされるかを決定します。`simple_majority` または `quorum_based` にすることができます。`consensus_mode = quorum_based` の場合、新しい設定 `difference_required_quorum`（0.0~1.0 の値、デフォルトは 0.5）が考慮されます。デフォルトは `consensus_mode = simple_majority` です。
- データ設定が異な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ヘルプ
新しい設定 `difference_tie_breakers` を追加しました。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` で定義し、`Initial Connection` のみを定義し、`node1` および `node2` を有効な BDR ノード名にする必要があります。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、接続セクション名前に「左」と「右」以外の接続名も使用できます。ユーザーが引き続き `Left Connection` と `Right Connection` を定義できるように下位互換性が維持されていますが、上記のように `difference_tie_breakers` の定義が必要な接続は 2 つだけです。以前は、「左」接続は常にタイブレーカーと見なされていました。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ていました。
- 多元比較では、PostgreSQL が存在する場合、接続の 1 つだけが PostgreSQL と異なることが必要です。
- 新しい設定 `stop_after_time` を実装して、数秒後に LiveCompare を手動で中断できるようにしました。デフォルトでは、`stop_after_time = 0` は LiveCompare が自動的に停止しないことを意味します（マニュアルの Ctrl-c によるのみ）。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 が手動または自動で中断されたかどうかに関係なく、コマンドラインで引数としてセッション ID を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 を、時間とエラー（存在する場合）に格納します。スクリプト `applied_*.sql` には、同じ適用済み DML も含まれるようになりました。
- 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 に対してのみ有効です。3.6.11 以降では、代わりにビルトイン関数 `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可能性がある問題を修正しました。
- CI：Ubuntu 18.04 LTS 向けに LiveCompare パッケージソフトもビルドされるようになりました。

2.1.24 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。
- 無視された相違：ユーザーは、出力 PostgreSQL 接続 (RM8939) で関数 `accept_divergence(session_id, table_name, difference_pk)` を手動で呼び出すことにより、特定の相違の相違の再チェックを停止できます。
- 揮発性の発散：一貫性のない行で `xmin` が差異の再チェックで変更された場合、LiveCompare は再チェックを停止し、差異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、相違を上書き (RM10964) としてマークします。
- 修正不可能な発散：自動修正が正常に適用された後、再確認時に `xmin` が変更されていなくても発散が残っている場合、LiveCompare は発散を修正不可能としてマークします (RM10964)。
- LiveCompare は、分岐がない場合はコード = 0 を返し、分岐がある場合は > 0 をリターンコード。
- すべての一般設定とともに、テーブルフィルターと行フィルターがテーブル `<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クリプトを開始する前に、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作成するとき、LiveCompare は宣言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 スクリプトを生成します。
- 自動修正が有効な場合 (`difference_mode = live_fix` または `difference_mode = offline_fix`)、正しい接続が PostgreSQL であることが必要です。
- セキュリティの問題を修正：テーブル/列名とすべてのリテラルが適切に引用されるようになりました (RM12530)。
- 予約語が列名として使用された場合の問題を修正 (RM12530)。
- ビュー `vw_differences` の列名の問題を修正 (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。

2.1.25 1.1.0 (2019-12-18)

BDR 3 のテーブルリストを作成する場合、LiveCompare は、左接続と右接続 (RT66502) の両方の BDR ノードに関連付けられているレプリケーションセットの積部分のみを考慮するようになりました。- パーティション化されたテーブルパーティションまたはパーティションが考慮されていた場合のテーブルリストの構築の問題を修正 (RT66499)。- 接続のバリデーション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改善しました。オラクルのエンコーディングおよび文字列ハンドリングに関する問題を修正しました。

2.1.26 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および使用できます。BDR バージョンが 3.6.11 以降の場合、LiveCompare では、BDR 関数 `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ために、`bdr_superuser` 権限を持つユーザまたは PostgreSQL スーパーユーザが必要です。BDR バージョンが 3.6.10 以前の場合、LiveCompare では、PostgreSQL 関数を使用し PostgreSQL レプリケーションオリジン管理を実行するために PostgreSQL スーパーユーザが必要です。それ以外の場合、LiveCompare はレプリケーション起点元を管理しようとしません。(RT66192 および RM11000)。
- LiveCompare は、BDR / pglogical 内の一貫性のない行からレプリケーションオリジン情報をフェッチできます。BDR バージョンが 3.6.11 以降の場合、LiveCompare では、`bdr_superuser` 権限を持つユーザまたは PostgreSQL スーパーユーザが行からレプリケーションオリジン情報をフェッチする必要があります。BDR バージョンが 3.6.10 以前または pglogical 3 が使用されている場合、LiveCompare では、`pglogical_superuser` パーミッションを持つユーザまたは PostgreSQL スーパーユーザが行からレプリケーションオリジン情報をフェッチする必要があります。それ以外の場合、LiveCompare はレプリケーションオリジン情報をフェッチしようとしません (RM11971)。
- 親テーブルが既にテーブルリストにある場合は、テーブルリストからパーティションを削除しテーブルリスト (RT65920 および RM10994)。

- pglogical および BDR のテーブルリストを作成するときは、レプリケーションセットテーブルを常に考慮してください (RT65920)。
- 空の文字列をハンドリングするときの問題を修正 (RT65918 および RT65988)。
- 0.11.0 から 0.12.0 へのアップグレード処理の問題を修正 (RT65918 および RT65988)。
- 最小 PK および最大 PK 値の決定の問題を修正しました。

2.1.27 0.12.0 (2019-11-01)

- `difference_fix_start_query` の設定のデフォルト値を変更して、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しました。これは、悪意のあるトリガーを作成することにより、データベースユーザーが LiveCompare で使用される特権ロールにアクセスできないようにオーダーに行われます。その結果、正しい接続のユーザは、ロールをテーブル所有者 (RM11000) にスイッチことができるニーズがあります。
- LiveCompare がテーブルリストを作成した後、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たり、スキーマの変更が破損したりするケースを処理しました。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します (RT65918)。
- `pg_replication_origin_session_setup` の実行時の問題を修正 (RT65988)。
- 最大 PK 値の決定の問題を修正 (RM11340)。
- データベースにラージのテーブルがある場合、テーブルからメタデータをフェッチするときのパフォーマンスの問題を修正しました。

2.1.28 0.11.0 (2019-10-10)

- 各差分チェックはテーブル `difference_log` に記録されるようになりました。これは、時間の経過とともに変化する各差分の分析に役立つ情報を提供します。それぞれの違いは、次のいずれかのステータスを通してできます。
- **検出済み (D)** : 違いが検出されたばかりです。`difference_mode = live_nofix` または `difference_mode = live_fix` の場合、LiveCompare は、結論が出るまで (以下の他のステータスを参照)、または最大 N 回 (`max_difference_check_attempts` の設定で構成可能)、各再チェックの間に X 秒待機するまで、差をマルチプル回再チェックします `min_time_between_difference_checks` の設定により構成可能)。 `difference_mode = offline_nofix` または `difference_mode = offline_fix` の場合、検出された各差異はすぐに **永続的** と見なされます。
- **永続 (P)** : `max_difference_check_attempts` の差を再チェックした後、LiveCompare は再チェックを停止し、差を **永続** としてマークします。 `difference_mode = offline_nofix` または `difference_mode = offline_fix` の場合、再チェックは行われなため、すべての違いは検出された時点で **永続的** としてマークされます。
- **不在 (A)** : `max_difference_check_attempts` に達する前に、LiveCompare が差異がもう存在しないことを検出した場合 (行は両方のデータベース間で一貫している)、LiveCompare は再チェックを停止し、差異を **Absent** としてマークします (以前のバージョンでは、LiveCompare は差分テーブルからレジスタを削除していました)。

- **許可されていません (N)** : 違いは検出されましたが、LiveCompare は自動的に修正できる違いの種類を制限しているため (ユーザ `difference_statements` および/または `difference_allow_null_updates` を介して) この違いを自動的に修正することは禁止されています。
- **修正済み (F)** : `difference_dml_right` フィールドから DML を適用することにより、Right Connection の LiveCompare によって差異が自動的に修正されました。
- **エラー (E)** : LiveCompare は、`difference_dml_right` のフィールドから DML を Right Connection に適用することで違いを修正しようとしたましたが、エラーが発生しました。エラーメッセージはフィールド `difference_dml_error` に記録されます。
- **出力接続の自動スキーマ変更** : ユーザが Output Connection を以前の LiveCompare バージョンで使用されていたデータベースに指定している場合、LiveCompare は自動的にスキーマ変更をハンドルします。0.11.0 では、LiveCompare はスキーマを再作成しますが、0.12.0 からは、実装はスキーマの変更のみを適用し、ユーザデータを保持します。
- **.ini ファイルからのセッション値を格納するためのテーブル connections および settings を作成しました。**
- **各差異の行表現形式の改善** : 行全体が JSON として保存されるようになりました。
- **xmin** を行の拡張列に格納します。
- 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 が改善されました。
- テーブルに複数行の文字列、null 値、または `bytea` 型の PK と列がない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 があったが、PK に前述の状況のいずれかがあった場合。

2.1.29 0.10 (2019-09-24)

- 新しい設定 `difference_mode` が追加されました。
- **offline_nofix** : 比較対象のテーブルにはロードがかかっていないため、差異は再チェックされません。違いは報告されますが、修正されません。
- **offline_fix** : 比較対象のテーブルにはロードがかかっていないため、差異は再チェックされません。違いは、見つかったときに Right Connection で報告および修正されます。
- **live_nofix** (デフォルト) : LiveCompare は、比較対象のテーブルにロードがかかっていると想定しているため、LiveCompare はそれらを再チェックして、最終的な整合性のためになくなっているかどうかを確認します。永続的な違いは報告されていますが、修正されていません。
- **live_fix** : LiveCompare は、比較対象のテーブルにロードがかかっていると想定しているため、LiveCompare はそれらを再チェックして、結果整合性のためになくなっているかどうかを確認します。永続的としてマークされている場合、差異は報告および修正されます。
- **difference_recheck** の設定は `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。

- `difference_mode = offline_nofix` または `difference_mode = live_nofix` の場合、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場合、DML スクリプトで LiveCompare によって生成される DML ステートメントの種類を制御する新しいグローバル設定 `difference_statements` が追加されました。`difference_statements` の値は次のとおりです。
 - `all` (デフォルト)
 - `inserts`
 - `updates`
 - `deletes`
 - `inserts_updates`
 - `inserts_deletes`
 - `updates_deletes`
- 新しいグローバル設定 `difference_allow_null_updates` (デフォルト `on`) が追加されました。これは、差分レポートまたは自動修正で `UPDATE SET col = NULL` などのコマンドを許可するかどうかを決定します。
- 新しいグローバル設定 `difference_fix_replication_origin` が追加されました。手動で設定されない場合、`pglogical 3` および/または `BDR 3` 比較のためにデフォルトで自動的に `bdr_livecompare` に設定されます。LiveCompare は、正しい接続に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、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、比較後の検証を可能にするために削除されませんが、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することに注意してください。
- 新しいグローバル設定 `difference_fix_start_query` が追加されました。これは各トランザクションの開始時に実行され、正しい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。LiveCompare は、`difference_fix_replication_origin` で指定されたレプリケーションオリジンを差分修正トランザクションが使用する `make` に、`difference_fix_start_query` も自動的に設定します。
- 新しい接続設定 `start_query` が追加されました。これは、接続が開かれるたびに任意のクエリーを実行に使用できます。
- 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。これは、コンソール出力にプログレスバーを表示するかどうかを決定します。バッチ実行に便利です。Postgres の比較では、検出されたそれぞれの差に、行の `ctid` も保存され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、検出された各差分には、行の `xmin` のレプリケーションオリジンも格納されます。
- 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

開始時に追加されます。

- 実行の終了時にグローバルプログレス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かったバグを修正しました。

2.1.30 0.9 (2019-08-30)

- 左または右接続でのオラクルデータベースのサポート。オラクル 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、オラクルに接続し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。PostgreSQL 以外の技術がいずれかの接続で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。ユーザが Postgres と Postgres の両方でハッシュの使用を無効にしたい場合、`full_comparison_mode = on` を構成ファイルで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ー処理できます。
- Left Connection と Right Connection の接続のいずれかが PostgreSQL にある場合、行ハッシュは `md5()` であるニーズがあります & lt; 11. それ以外の場合、両方の接続で `hashtextextended()` が使用されます。これにより、混在 PostgreSQL バージョンの比較が可能になります（例、9.4 と 12）。
- `logical_replication_mode` を設定すると、Initial Connection および Left Connection に PostgreSQL のバージョンと拡張機能の存在がバリデーションされます。テーブルリストは Left Connection から作成され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、混在テクノロジーの比較が可能になります（PostgreSQL とオラクル、BDR と PostgreSQL、BDR 2 と BDR 3 など）。
- ユーザーが差分の再確認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`）にログオン。
- 後の分析のために、左と右の接続情報がレポートデータベース（テーブル `sessions`）にログオンしています。
- テーブルに重複した行があるコーナーケースでの差分チェックのバグを修正しました。
- テーブルの正しい接続にさらに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レポートのバグを修正しました。

2.1.31 0.8 (2019-08-07)

- テーブルにマルチプルのフィールドを持つ PK がある場合の単行のフェッチ（不整合をチェックするため）を修正しました。

2.1.32 0.7 (2019-08-06)

- 行のないテーブルのハンドリングを修正しました。
- 空のセクションのハンドリングが改善されました。

2.1.33 0.6 (2019-06-17)

- ロギングコンポーネントを変更しました。

2.1.34 0.5 (2019-06-17)

- バグ修正：
- 順序付け演算子を使用せずにデータ型を並べ替えするサポート。
- `hashtextextended()` が使用できない場合（PG & lt; = 10）、`md5()` をレコードハッシュとして使用します。
- 構成ファイルが存在しない場合、適切なメッセージします。

2.1.35 0.4 (2019-06-12)

- 2ndQuadrant CI パイプラインに含めるための準備。

2.1.36 0.3 (2019-05-29)

- 接続を指定する DSN のサポート。
- テーブルおよび行フィルターの改善。
- さまざまなタイプのロジカルレプリケーションのサポート：
- ネイティブロジカルレプリケーション
- pglogical
- bdr BDR サポート
- ユーザが接続のノード名を指定できるようにします。

- ユーザ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。

2.1.37 0.2 (2019-05-21)

- ハッシュの改善：md5() の代わりに `hashtext()` と `hashtextextended()` を使用。
- フェッチは準備済みステートメントを使用して実行されます。

2.1.38 0.1 (2019-05-17)

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

2.2 Requirements

LiveCompare の要件：

Python 3.6 または 3.7 PostgreSQL / EDB Postgres Extended 9.5+ / EPAS 11+ (出力接続上) PostgreSQL / EDB Postgres Extended 9.4+ / EPAS 11+ または オラクル 11g + (比較対象のデータ接続上)

LiveCompare には、Debian 10 +、Ubuntu 16.04+、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、次を参照してください。

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

LiveCompare は以下の上にインストールされます。

- 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 python3 パッケージソフト。または
- python-36 パッケージソフトで提供される CentOS / RHEL 7 用の Python 3.6。

CentOS / RHEL ディストリビューションでは、LiveCompare には 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の `pip` または `pip3` を使用して、最新の `tqdm` をインストールすることができます。

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

2.2.1 TPAexec との LiveCompare

TPAexec の次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
  location: main
  node: 2
  role: replica
  upstream: livem1node1
  published_ports:
    - 5402:5432
```

TPAexec の詳細については、次を参照してください。

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

2.3 Supported Technologies

LiveCompare は、PostgreSQL、BDR、オラクル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 に関するメタデータをフェッチするために使用されます接続。データ接続が pglogical または BDR であり、かつ replication_sets または node_name 設定が使用されます。が必要です logical_replication_mode = pglogical または logical_replication_mode = bdr。それ 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**：ツールが実行するために接続する実際のデータベースコネクションデータ比較。リストの最初の接続は、「テーブル FilterandRow Filter、および Initial BDR ノードに関する情報を収集するための接続。もし logical_replication_mode = bdr および all_bdr_nodes = on、そして LiveCompare は「初期」と同じ BDR クラスターのパートであるすべての BDR ノードを検討する 接続。この場合、データ接続を定義する必要はありません個別に。比較として、修正はすべてのデータ接続に潜在的に適用できます。そして、コンセンサスの決定は行ごとに機能します。
- **出力**（必須）：LiveCompare は次のスキーマを作成します livecompare、いくつかのテーブルとビュー。これは、進捗を維持するために必要です比較セッションに関するデータの報告。PostgreSQL または 2ndQPostgres 接続。

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について説明します。

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

2.3.1 PgBouncer のサポート

LiveCompare は PgBouncer を介してノードに対して使用できますが、LiveCompare は PostgreSQL でプリペアドステートメントを使用するため、`pool_mode=session` を使用する場合に限りです。`pool_mode` が `transaction` または `statement` の場合は不可能です。

2.4 Command-line Usage

2.4.1 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（以下のセクション `Settings` を参照）。

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

livecompare my_project.ini
```

LiveCompare の実行中に、`N+1` プログレスバーが表示されます。`N` はプロセスの数です（設定でプロセスの数を指定できます）。最初のプログレスバーは実行全体を示し、他のプログレスバーは特定のプロセスで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、左から順に次のとおりです。

- プロセスの数
- テーブル名前
- ステータス。比較ラウンドの ID の後に現在の ID テーブルチャンク（`p1/1` はテーブルが分割されなかったことを意味します）。ステータスにある場合 `setup`、テーブルが分析されていることを意味します（行数のチェックと分割必要に応じて）
- 処理された行数
- この比較ラウンドで考慮される合計行の数
- 経過時間
- 完了までの推定時間
- 1 秒あたりの記録の速度

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、ハッシュファンクションを使用して、どのジョブが行を考慮するかを決定します。これにより、比較が個別に遅くなる可能性があります。比較全体としては、指定されたテーブルの並列処理の恩恵を受ける可能性があります。

プログラムの実行中に、`Ctrl-c` を押すと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、ソフトストップで停止できます。livemaster.pid 名前付けのファイルで、セッションフォルダー（例では lc_session_6）内のマスタープロセスの PID を保持します。その後、kill -2 <PID> を呼び出してソフトストップできます。

その後、いつでも以前にキャンセルしたセッションを再開でき例。

```
livecompare my_project.ini 6
```

プログラムが終了し、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.
```

しかし、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:
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.
```

(次のページに続く)

```
Saved file lc_session_6/differences_20200129.out with the list of differences per
→table.
```

You can also get a list of differences per table with:

```
select * from livecompare.vw_differences where session_id = 6;
```

Too see more details on how LiveCompare determined the differences:

```
select * from livecompare.vw_consensus where session_id = 6;
```

```
Script lc_session_6/apply_on_the_first_20200129.sql was generated, which can be
→applied to the first connection and make it consistent with the majority of
→connections.
```

You can also get this script with:

```
select difference_fix_dml from livecompare.vw_difference_fix where session_id = 6 and
→connection_id = first;
```

2.4.2 再確認モード

BDR 環境では、BDR が検出した相違は、最終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、後で存在しなくなる可能性があります。いくつかの要因に応じて、レプリケーションラグにより LiveCompare が誤検知をレポートする可能性があります。

これを克服するために、レプリケーションラグが減少した後、またはデータが既に追いついた後、ユーザーは手動で以前に見つかった違いのみで再チェックを実行できます。この実行モードは「recheck」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 recheck ログを生成し、lc_session_X ディレクトリにすでに存在するすべてのレポートを更新します。

重要：compare を再開する場合、または recheck で実行する場合、LiveCompare は設定および接続属性がセッションの作成時と同じかどうかを確認します。相違が見つかった場合、適切なメッセージで実行を終了します。

2.4.3 競合モード

LiveCompare を conflicts モードで実行するには、次のコマンドで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 サポートの章をご覧ください。

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を表示します。各テーブルについて、LiveCompare がテーブルの処理にかかった時間を示します。行の合計数と処理された行数、数テーブルで違いが見つかり、無視される列の最大数も見つかりました。もしあれば。

完全な要約を取得するには、出力データベースに対して次のクエリー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、このファイルそれぞれの違いに関する有用な情報を示します。次の場合、このファイルは生成されません。違いはありません。

例、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ー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を分岐させるかを決定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、これファイルには **first** データベースに適用される DML コマンドが表示され make。 **first** データベースは他のすべてのデータベースと一貫しています。例、上記の違いにより、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;
UPDATE public.categories SET categoryname = $lc1$Games Changed$lc1$ WHERE (category) = 10;
INSERT INTO public.categories (category,categoryname) VALUES (17, $lc1$Test 1$lc1$);
INSERT INTO public.categories (category,categoryname) VALUES (18, $lc1$Test 2$lc1$);

COMMIT;
```

LiveCompare はこのスクリプトを自動的に生成します。 **first** データベースの不整合を修正オーダーには、**first** データベースでスクリプトを実行だけです。

LiveCompare は、一貫性のないデータを持つ各データベースに対して同様の `apply_on_*.sql` スクリプトを生成します。

2.5.1 比較中絶

比較セッションを開始する前に、LiveCompare はすべての接続を試行します。到達可能な接続の数が2つ以上でない場合、LiveCompare は適切なエラーメッセージで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。健全性チェックのいずれかが失敗した場合、LiveCompare は適切なエラーメッセージで比較を中止します。

すべての到達可能な接続で使用可能なテーブルを考慮して、LiveCompare は比較対象のテーブルのリストを作成し、Table Filter もアカウントします。特定のテーブルが少なくとも2つの接続上に存在しない場合、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。このステップは「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（例、ユーザが特定のテーブルにアクセスできない場合）、「セットアップエラー」と呼ばれます。`abort_on_setup_error` が有効な場合、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、エラーのある特定のテーブルのみがテーブル比較を中止し、LiveCompare は次のテーブルに移動します。

LiveCompare はテーブルごとにテーブル比較を開始し、最初に LiveCompare は到達可能なすべての接続のテーブル定義をチェックします。テーブルに同じ列と列データ型がない場合、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。

2.5.2 比較キー

比較される各テーブルについて、テーブル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 **Comparison Key** を構築します。

1. ユーザが構成した場合、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、少ない列を持つ **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（**NULL** 列は、`ignore_nullable = false` の場合も考慮できます）。

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

2.5.3 修正する違い

LiveCompare は、次の相違点を特定して修正を提供できます。

- データ接続の大部分に行が存在します。修正は異なるデータベースの **INSERT**。
- データ接続の大部分には行が存在しません。修正は異なるデータベースの **DELETE**。
- 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** である。

デフォルトでは、`difference_statements = all`、つまり、LiveCompare は、検出した各差異に対して 3 つの DML タイプ (**INSERT**、**UPDATE**、**DELETE**) をすべて適用しようとします。ただし、`difference_statements` の設定値を変更することにより、差分修正を提供するときに考慮する DML LiveCompare のタイプを指定できます。

- `all` (デフォルト) : **INSERTs**、**UPDATEs**、および **DELETEs** を修正。
- `inserts` : **INSERTs** のみを修正。
- `updates` : **UPDATEs** のみを修正。
- `deletes` : **DELETEs** のみを修正。
- `inserts_updates` : **INSERTs** および **UPDATEs** のみを修正。
- `inserts_deletes` : **INSERTs** および **DELETEs** のみを修正。
- `updates_deletes` : **UPDATEs** および **DELETEs** のみを修正。

`difference_statements` の値が `all`、`updates`、`inserts_updates`、または `updates_deletes` である場合、**NULL** を列に設定する **UPDATE** を無視するよう LiveCompare に指示することができます。

2.5.4 差分ログ

テーブル `difference_log` は、LiveCompare がチェックするたびに差異に関するすべての情報を保存します。ユーザーは LiveCompare を再チェックモードでマルチプル回実行できるため、この表は、LiveCompare が再チェックを行っていた時間枠で差異がどのように変化したかを示しています。

- **検出済み (D)** : 違いが検出されたばかりです。再確認および修正モードでは、LiveCompare は、すべてのパーマネントとタイの違いを検出済みとしてマークし、それらを再確認してください。
- **パーマネント (P)** : データがまだ残っている場合、差異を再チェックした後 LiveCompare は相違点を **Permanent** としてマークします。
- **Tie (T)** : Permanent と同じですが、決定するための十分なコンセンサスがありませんとの接続が大半です。
- **不在 (A)** : 再チェック時に、LiveCompare が違いを発見した場合もう存在しない（行は両方のデータベース間で一貫している）、その後 LiveCompare は、差異を **不在** としてマークします。
- **Volatile (V)** : 再チェック時に、一貫性のない状態で `xmin` が変更された場合行、次に LiveCompare はその差を **Volatile** としてマークします。
- **無視 (I)** : ユーザーは、特定の差異の差異再チェックを停止できます。手動でファンクションを呼び出す `<livecompare_schema_name>.accept_divergence(session_id, table_name, difference_pk)` 出力 PostgreSQL 接続で。例：

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

2.6 BDR Support

LiveCompare は、BDR ノードおよび非 BDR ノードに対して使用できます。

`logical_replication_mode = bdr` を設定 make と、比較するすべてのデータベースが同じ BDR クラスターに属しているとツールが想定します。次に、ノード名を接続として指定し、レプリケーションセットを指定してテーブルをフィルタリングできます。

例、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。最初にこのノード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。

ここで、3つの BDR ノードを比較することを検討してください。LiveCompare は **Initial Connection** から始まる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ついて BDR メタデータを検索し、ノードに接続します。

ノートが 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、`postgres` ユーザとして、`Initial Connection` 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、テーブルリストを作成します。

例、`.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
```

LiveCompare に 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 ノードのリストをフェッチします。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。例、移行プロジェクトで役立つ単一の Postgres 接続に対して BDR クラスター全体を比較することができます。

[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=olddpg 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 の代わりにノートメタデータフェッチを有効にするには、`logical_replication_mode = bdr` ではなく `logical_replication_mode = pglogical` を設定するだけです。

2.6.1 BDR 監視ノード

BDR のレプリケーションセットを使用して、特定のテーブルを BDR レプリケーションに含めるように構成し、テーブルが属するレプリケーションセットに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することもできます。これにより、BDR シャーディングや BDR ウィットネスノードの使用など、さまざまなアーキテクチャが可能になります。

BDR ウィットネスは、他のノードから DML を複製しない通常の BDR ノードです。証人の目的は、ラフトコンセンサス投票で定足数を提供することです（BDR 証人ノードの詳細については、BDR の文書を参

照してください)。レプリケーションセットの構成方法に応じて、Witness は DDL をレプリケートする場合とレプリケートしない場合があります。つまり、BDR 証人には次の 2 つのタイプがあります。

- 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 を複製するノード。したがって、空のテーブルがあります。

最初のケースでは、BDR Witness が比較に含まれていても（手動で [Connections] の下で、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、BDR ウィットネスにテーブルが存在します。しかし、証人に欠けているデータを分岐としてレポート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チェックします。

- ノードがサブスクライブ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。
- レプリケーションセット（存在する場合）フィルター `replication_sets` で定義されたユーザ Table Filter の下。

レプリケーションセットの 3 つのリストすべての積部分が空の場合 (BDR ウィットネスの場合)、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行われ、テーブルは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ます。

2.6.2 BDR クラスターの違い

LiveCompare はローカルノードのみに変更を加えます。修正の変更が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（レプリケーションオリジンの名前は、`difference_fix_replication_origin` の設定を調整することで構成できます）。`bdr_local_only_origin` というレプリケーションオリジンがまだ存在しない場合、LiveCompare はすべての BDR 接続でそれを作成します。

重要：BDR 3.6.18 では、ローカルのみトランザクションの適用に使用され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が導入されていることにノートしてください。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、以下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を使用するように現在のトランザクションを設定します `bdr_local_only_origin`。したがって、実行される DML には `xmin` が関連付けられます。
`bdr_local_only_origin`;
- 現在のトランザクションの日時を過去に設定します。BDR は、データベース、LiveCompare DML で実行されている実際の DML と競合します。常に対立を失います。

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` を任意の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くなり可視。)

LiveCompare がメタデータを適切にフェッチオーダーには、少なくとも `bdr_superuser` 権限を持つ PostgreSQL ユーザが必要であることにノートしてください。

PostgreSQL ユーザが `bdr_superuser` または PostgreSQL のスーパーユーザ権限を持っている場合、レプリケーション起点元に関する上記のすべての手順は出力スクリプトにのみ適用されます。それ以外の場合、LiveCompare は、レプリケーションオリジンを関連付けずに修正を生成します (`SET LOCAL bdr.xact_replication = off` を使用すると、トランザクションレプリケーションは引き続き無効になります)。ただし、DML スクリプトを適用する場合は、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、競合の解決に関して LiveCompare が通常のユーザアプリケーションと同じ優先順位を持つことになります。また、修正に関連付けられたレプリケーションオリジンがないため、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ーは使用できません。

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()` : LiveCompare に よ っ て 実 行 さ れ 、 `difference_fix_replication_origin` で指定されたレプリケーションオリジン (デフォルトでこのレプリケーションオリジンが存在しない場合、`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、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ています `difference_fix_replication_origin`;
- `bdr.difference_fix_xact_set_avoid_conflict()` : 生成された DML に含まれますスクリプトにより、トランザクションは過去 (2010-01-01) に設定されるため、修正 LiveCompare によって適用されたトランザクションは、競合がある場合、常に失われます。

上記の関数には、PostgreSQL スーパーユーザではなく `bdr_superuser` が必要です。BDR 3.7.0 以降、これらの機能は廃止されました。LiveCompare は、PostgreSQL スーパーユーザ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以下のみを含めます。

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

2.6.3 BDR の競合

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

`compare` モードはテーブルのすべてのコンテンツを全体として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/テーブルにのみフォーカスします 3.7 および BDR 4。

とはいえ、`conflicts` 実行モードは、特定のテーブルから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と予想されます。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、BDR 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、自動競合解決によりノード間でタプルが発散することが知られているいくつかのケースがあります。したがって、LiveCompare の `conflicts` 実行モードは、時間と結果のバランスが取れた状態で、整合性の確認と保証にヘルプします。

競合の例-----

`node3` で次のクエリーを実行ことを想像してください。

```
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` で次の競合を確認できます。

```

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 に到着したときに、tbl の表に b = 3 の行がないことを意味します。ただし、INSERT は node1 から node3 に後で到着し、b = 3 の行を node3 に追加した可能性があります。これが、node3 の現在のシチュエーションです。

```

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

```

BDR クラスターは分岐しています。

このような相違を検出して修正するオーダーに、LiveCompare を compare モードで実行できますが、比較セットのサイズに応じて（イメージテーブル tbl が非常にラージ）、長時間、さらには数時間かかる場合があります。

これは、conflicts モードが役立つシチュエーションです。この場合、delete_missing の競合は可視からのみ表示されますが、LiveCompare は競合のログに記録された行（key_tuple、local_tuple、remote_tuple および apply_tuple）から PK 値を抽出し、影響を受けるテーブルでのみ自動クラスタ全体の比較を実行できます。すでに PK 値でフィルタリングしています。次に、比較により、クラスター内のすべてのノードの現在の行バージョンがチェックされます。

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

```

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

[Initial Connection]

```

(次のページに続く)

(前のページからの続き)

```
dsn = 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 が現在の比較セッションの番号) 内で、LiveCompare はファイル conflicts_DAY.out (ファイル名前の DAY を現在の日付に置き換えます) を書き込み、すべての分岐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、例次のクエリーを使用して、競合に関する詳細を確認できます。

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

次のようなものが表示されます。

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
key_tuple	{"a": null, "b": 3, "c": null}
local_tuple	
remote_tuple	
apply_tuple	
conflict_type	delete_missing
conflict_resolution	skip

(次のページに続く)

(前のページからの続き)

```

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分岐しているノードを見つけたことを意味します。ビュー `livecompare.vw_conflicts` は、非分岐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` を自動的に生成します（ファイル名前の `DAY` を現在の日付に置き換えます）。

```

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;

```

LiveCompare は、他の 2 つのノードに行が存在しないため、`DELETE` に `b = 3` が `node3` からの行にあることを提案しています。デフォルトでは、LiveCompare は大部分のノード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 クラスターの整合性が再び取れます。

`--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、競合の解決によりクラスター全体の一貫性が保証されるように、`--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オーダーには、`bdr_superuser` または PostgreSQL スーパーユーザのユーザで LiveCompare を実行する必要がありますことにノートしてください。

競合フィルター-----~~~~

LiveCompare に、`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指示することもできます。例：

```

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

```

2.6.4 ミキシング技術

`node_name` および `replication_sets` のメタデータは `Initial Connection` でフェッチされることにノートしてください。したがって、それは `pglogical` および/または `BDR` 対応のデータベースでなければなりません。

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

混在テクノロジーの比較を実行することは可能です、例：

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

2.7 Oracle Support

LiveCompare を使用して、オラクル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。

例、データ接続で `technology = oracle` を定義できます。他の設定を使用して、オラクル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。

オラクルデータベースと PostgreSQL データベースのシンプル比較例を次に示します。

```
[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
```

(次のページに続く)

```
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、オラクルの `schema_name` はユーザ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ています：

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

`schema_qualified_table_names = off` を設定することにより、スキーマで修飾された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。つまり、LiveCompare は、接続されているオラクルユーザに属するテーブルのみを検索します。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、ロールまたは設定でデフォルトの `search_path` を設定する必要があります。または、接続 `start_query` パラメータを使用して、適切な例を設定できます。

```
[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

[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
```

(前のページからの続き)

[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ノートしてください。

BDR データベースをオラクルと比較する必要がある、Initial Connection、node_name および replication_sets 機能（前述）を利用したい場合は、次のようにオラクルへの最後のデータ接続を指定できます。

[General Settings]

```
logical_replication_mode = bdr
max_parallel_workers = 4
oracle_user_tables_only = on
oracle_ignore_unsorttable = 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
```

例、BDR クラスター全体を単一のオラクルデータベースと比較することもできます。

[General Settings]

```
logical_replication_mode = bdr
max_parallel_workers = 4
oracle_user_tables_only = on
oracle_ignore_unsortable = on
column_intersection = on
force_collate = C
difference_tie_breakers = oracle
all_bdr_nodes = on
```

[Initial Connection]

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

[Oracle Connection]

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

[Output Connection]

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

[Table Filter]

```
replication_sets = set_name = bdrgroup
```

2.7.1 要件

LiveCompare は、追加のソフトウェアをインストールする必要なく、すぐに使用できる PostgreSQL データベースで動作することにノートしてください。

ただし、オラクルに接続できるオーダーにするには、LiveCompare に追加のソフトウェアが必要です。

オラクル Instant Client

オラクル 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、使用するオペレーティングシステムに応じて特定のフォルダに抽出します）。

- **MacOSX** : オラクル Instant Client のダウンロード 64-bit) ~/lib で抽出します。
- **Linux** : オラクル Instant Client のダウンロード 32-bit) 64-bit)

それをシステムにインストールしてから、LD_LIBRARY_PATH を設定します。

- **Windows** : オラクル Instant Client のダウンロード (32-bit) (64-bit) それを LiveCompare フォルダーに抽出します。

cx_Oracle Python モジュール

Python モジュール *cx_Oracle* は、LiveCompare がオラクルデータベースに接続できるように、システムにインストールして使用可能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 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で実行することが推奨されているため、次のコマンドを使用して、postgres ユーザに対してのみ PIP を介して cx_Oracle モジュールをインストールできます。

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

2.7.2 違い

LiveCompare で違いが見つかった場合、PostgreSQL 接続にのみ適用される DML スクリプトが生成されます。オラクル接続に適用される DML スクリプトは生成されません。

2.7.3 BLOB および CLOB データ型

LiveCompare は、PostgreSQL の同等のフィールドが text 型である場合、オラクルの CLOB フィールドを比較できます。同じことがオラクルの BLOB フィールドにも当てはまります。PostgreSQL の同等のフィールドは PostgreSQL 型でなければなりません。

ただし、デフォルトでは、LiveCompare は BLOB / CLOB フィールドがプライマリキーにある場合、またはテーブルにプライマリキーがない場合、BLOB / CLOB フィールドをハンドルしません。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
```

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

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

2.7.4 互換性のない照合

オラクルでは、通常、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
```

つまり、NLS_LANG およびその他の言語設定に関係なく、オラクルのすべての ORDER BY 操作は文字バイナリコードを使用して実行されます。

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 LiveCompare は PK 値を並べ替えときに問題を検出する可能性があり、誤検出につながる可能性があります。

これを回避するには、Postgres で照合順序（たとえば、C 照合順序）を強制することができます。これにより、オラクルの同じソート動作と一致します。

```
force_collate = C
```

LiveCompare は、比較セッションにオラクルと PostgreSQL が関係していることを検出した場合、ユーザが他の値を設定していない限り、LiveCompare はすでに force_collate = C を設定します。

2.7.5 共通ハッシュ

デフォルトでは、PostgreSQL とオラクルを比較する場合でも、LiveCompare には comparison_algorithm = block_hash があります。ただし、これらのルールに従って「共通ハッシュ」が構築されます。

- すべての列の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 データ型が処理されてオラクルを模倣します。
- この方法では、完全な行表現形式が両側で 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る場合、LiveCompare は自動的に Posted の場合と同様に、Postgres に、次に full_row にフォールバックします。Postgres の比較。
- Oracle の BLOB、CLOB、NCLOB フィールドは、最初の 2000 文字に制限されてオラクルのみ (comparison_algorithm = full_row は BLOB と CLOB 全体の比較を許可します)；オラクルでは、行全体の表現形式が 4000 文字を超えてはなりません。

行全体の表現形式が 4000 文字を超える場合、LiveCompare はその特定のテーブルの比較をアボートし、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
```

それ以降の LiveCompare バージョンは、これらの特定のテーブルの 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、これらのテーブルのみで個別の比較セッションを構成することです。

Common Hash は、オラクル 12c 以降で `standard_hash` ファンクションを使用します。オラクル 11g では、`standard_hash` ファンクションは使用できないため、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せんが、オラクル側のユーザに追加の権限が必要になる場合が例ます。

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

2.8 Settings

2.8.1 一般設定

- `logical_replication_mode`: プログラムが接続を解釈する方法に影響し、テーブルフィルターの設定（以下の詳細を参照）、および必要な要件比較を開始する前に接続を確認してください。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（下記参照）のみを指定して、LiveCompare 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作成します。デフォルト: `off`。
- `max_parallel_workers`: 考慮される並列プロセスの数。それぞれ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 $N + 1$ のオープン接続を保持します。各データ接続に 1 つ、出力データベースに 1 つです。

- `buffer_size`: すべてのデータでテーブルから取得される行の数フェッチオペレーション。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`: データフェッチに準備済みステートメント (`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) のみがフェッチされるため、3つのモードすべてのインパクトが最も小さく、同じ理由により安全なフェッチモード。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、このフェッチメソッドは優れたパフォーマンスを提供しますが、ラージテーブルではパフォーマンスの低下が感じられます。これはデフォルトであり、サーバーのロードが中～高の場合に強く推奨されます。

-`server_side_cursors_with_hold`: サーバサイドのカーソル `WITH HOLD` を使用してデータをフェッチします。テーブルデータが単一のトランザクションで取得されると、`xmin` が抑制され、膨張とレプリケーションの問題が発生し、`VACUUM` が正常に実行されないことがあります。また、`WITH HOLD` 句は、Postgres にクエリーをマテリアライズするように指示します (データが具体化されるまで数秒間ハングする場合があります)。したがって、テーブルデータ全体が RAM を消費し、Postgres 側のディスクに一時ファイルとして保存できます。そのインパクトはすべて `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` を少し増やすことで速度を改善できます。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、このフェッチメソッドは `prepared_statements` と比べて利点はありませんが、マルチプル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例でロードが非常に低い場合にのみお勧めします。

-`server_side_cursors_without_hold`: データのフェッチにサーバサイドカーソル `WITHOUT HOLD` を使用します。`server_side_cursors_with_hold` のように、このモードは `xmin` を抑制できるため、Postgres で膨張、`VACUUM`、レプリケーションの問題を引き起こす可能性があります。 `WITHOUT HOLD` カーソルは比較セッション全体でオープントランザクションを必要とするため、このようなインパクトは大きくなります (これは今後のバージョンで廃止されます)。snapshot 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。クエリー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くなります。非同期データのフェッチは許可されていません。パフォーマンスの観点では、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、ラージテーブルの場合は速くなります。データベースのロードが中程度の場合に推奨されます。

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。プリペアドステートメントを使用すると、データベースサーバ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により、サーバサイドが常に最新バージョンの行を見ることがデータベースことです。そのため、稼動の高ロードサーバーには `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、低ロードサーバーのいずれかの `server_side_cursors_*` 設定。最適な戦略は、おそらく非常にラージテーブルの場合は `server_side_cursors_without_hold` を、残りのテーブルの場合は `prepared_statements` を混在させることです。コスト/便益比の比較については、以下の表を参照してください。

	pre-prepared_statements	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

オラクルに関する注意：オラクルの場合、`data_fetch_mode` 設定は完全に無視され、準備されたステートメントやカーソルを使用せずに、`LIMIT` で直接クエリを使用してデータが常にオラクルからフェッチされます。

- `parallel_chunk_rows`：分割を検討するために必要な最小行数並列比較のためにテーブルをマルチプルのチャンクに分割します。フェッチに使用されるハッシュデータなので、労働者は互いに衝突しません。各テーブルチャンクにはこれ以上ありません `parallel_chunk_rows` 行より。任意の値に設定 & lt; 1 はテーブルを無効にします 分割。デフォルト：0（無効）。

重要：テーブルの分割はラージテーブルをマルチプルのワーカーで並列に比較するのにヘルプしますが、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` は最新のハードドライブでは保守的すぎると考えられる）に応じて、Postgres クエリープランナはビットマップヒープスキャンを誤って優先し、データベースが SSD で実行されている場合、ビットマップヒープスキャンを無効にします LiveCompare は、比較のパフォーマンスを大幅に向上させることができます。これは、接続ごとに `start_query` 設定を使用して実行できます。

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

- `parallel_data_fetch`：データフェッチを並行して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）。多方向のパフォーマンスを改善比較。データ接続が PostgreSQL でない場合、この設定は自動的に無効になります。次の場合にのみ許可されます `data_fetch_mode = prepared_statements` または `data_fetch_mode = server_side_cursors_with_hold`。デフォルト：on。
- `comparison_algorithm`：テーブル行を介した LiveCompare の動作に影響しますデータを比較します。ハッシュの使用は、行全体の比較よりも高速です。想定できるもの次の値のうち：

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

```
- `row_hash`: Enables row comparison using hashes and enables table splitting. Tables are split so each worker compares a maximum of `parallel_chunk_rows` per table. Data row is hashed in PostgreSQL, so the comparison is faster than `full_row`. However, if for a specific row the hash does not match, then for that specific row, LiveCompare will fallback to `full_row` algorithm (i.e., compare row by row). If any data connections
```

(次のページに続く)

is not PostgreSQL, then LiveCompare uses a row hash that's 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`: ロギングするまで待機する時間 (秒) ログへの「ハートビート」メッセージ。各ワーカーはラウンドごとに個別に追跡します比較されるパート。デフォルト: 30 秒。
- `min_time_between_round_saves`: それぞれを更新前に待機する時間 (秒) 比較アルゴリズムが進行中のラウンド状態。ラウンドセーブはハートビート中に発生するため、`min_time_between_round_saves` は大きくする必要があります `min_time_between_heart_beats` 以上。ラウンド時に終了すると、LiveCompare は常にそのテーブルのラウンド状態を更新します。デフォルト: 60 秒。

重要: ユーザが 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、各ワーカーの行数に対応 `comparison_cost_delay` 秒の仮眠を取る前に処理されます。デフォルト 0 に設定すると、各ワーカーは昼寝をせずに行を処理します。
- `comparison_cost_delay`: `comparison_cost_limit` > 0 の場合、この設定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`: LiveCompare が自動的に開始するまでの時間 (秒) ユーザが Ctrl-c にヒットしたかのように自分自身を停止します。あった比較セッション中断され、まだ終了していない場合は、セッションを渡すことで再開できますコマンドラインの引数としての ID。デフォルトは `stop_after_time` = 0 です。自動割り込みが無効であることを意味します。
- `consensus_mode`: LiveCompare が使用する合意アルゴリズムデータ接続は異なります。可能な値は `simple_majority` です。 `quorum_based` または `source_of_truth`。 `consensus_mode` = `source_of_truth` の場合 `difference_sources_of_truth` を埋める必要があります。デフォルトは `simple_majority` です。
- `difference_required_quorum`: `consensus_mode` = `quorum_based` の場合、これ指定された設定では、どの接続を決定するために最小クォーラムが必要です発散。0.0 から 1.0 の間の数値である必要があります (0.0 は接続がないことを意味します 1.0 はすべての接続が必要であることを意味しますが、どちらの場合も極端です使用しないでください)。デフォルト値は 0.5 です。それに近い値。
- `difference_sources_of_truth`: 接続名のコンマ区切りリスト (またはノード名 (`logical_replication_mode` = `bdr` および `all_bdr_nodes` = `on` の場合) 真実のソースと考えるべきです。 `consensus_mode` = の場合にのみ使用されます `source_of_truth`。 For example: `difference_sources_of_truth` = `node1`, `node2`。でこの例では、セクション `node1 Connection`

および node2 Connection.ini ファイルまたは all_bdr_nodes = on で定義し、「Initial Connection is defined, while node1 and node2」は有効な BDR ノードである必要があります名前。

- **difference_tie_breakers**: 接続名 (または ノード) のコンマ区切りリスト名前 (logical_replication_mode = bdr および all_bdr_nodes = on の場合) コンセンサスアルゴリズムがタイを見つけたときはいつでもタイブレーカーと見なされるシチュエーション例: difference_tie_breakers = node1,node2。これで例、セクション node1 Connection および node2 Connections のいずれか.ini ファイルまたは all_bdr_nodes = on で定義され、Initial Connection is defined, while node1 and node2 は有効な BDR ノードである必要があります名前。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**: どんな種類の DML ステートメントになるかを制御します LiveCompare によって生成されます。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」などのコマンドを決定します col = NULL は差分レポートで許可されます。デフォルト: on`。
- **difference_statement_order**: DML ステートメントのオーダーを制御します LiveCompare によって生成されます。difference_statement_order の値 次のいずれかになります。

```
- `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、LiveCompare が作成しない場合、特定のレプリケーションオリジンが作成されますまだ存在している場合は、レプリケーションオリジンを使用して DML で適用スクリプトを作成します修正。設定名前は、LiveCompare が使用するレプリケーションオリジン。ユーザが値を設定しない場合この設定の場合、LiveCompare は自動的に設定されます difference_fix_replication_origin = bdr_local_only_origin。ことに注意してください LiveCompare が作成レプリケーションオリジンは、検証を許可するために削除されません比較後、必要に応じて、レプリケーションオリジンを手動で設定できます後で落とした。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の適用に使用され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が導入されていることにノートしてください。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。ユーザがこのレプリケーションオリジンを削除しようとしなことをお勧めします。

- **difference_fix_start_query**: で実行される任意のクエリー LiveCompare によって生成された適

用スクリプトの始まり。さらに、BDR 比較実行中で、`difference_fix_start_query` が空の場合、LiveCompare は次のことも自動的に行います。

```
- 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`: 進行状況バーを表示するかどうかを決定しますコンソール出力。この設定を無効にすると、バッチに役立ちます。実行。デフォルト: `on`。
- `output_schema`: 出力接続では、比較するスキーマ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、特定の列が含まれます。行のすべての実際の列のハッシュ。この設定は、この列。デフォルト: `livecompare_hash`。
- `rownumber_column_name`: 一部のフェッチでは、`row_number()` ファンクションを使用する必要がありますクエリー列内の値。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 はデバッグ行に役立つ可能性のある各分岐行のオリジン名前目的。デフォルト: `off`。有効にするには、`logical_replication_mode` セットが必要です `pglogical` または `bdr` へ。
- `column_intersection`: この設定が有効な場合、特定のテーブルに対して比較されている場合、LiveCompare は次の列の積部分でのみ機能します。すべての接続のテーブル。次のいずれかに存在する可能性のある余分な列を無視します。接続。この設定を無効にすると、LiveCompare は次のことを確認します。列はすべての接続のテーブルで同等であり、比較をアボートします列の不一致がある場合のテーブルの。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 列を異なるものにするとはできません。

- `ignore_nullable`: 特定のテーブル比較のために、LiveCompare が使用している場合主キーとは異なる比較キー、それから LiveCompare はすべてを必要とします `ignore_nullable` が有効な場合、列は `NOT NULL` になります (デフォルト)。それは `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。LiveCompare が比較で `null` 値を許可する列を考慮できるようにします。コーナーケースは誤検知を引き起こす可能性があります。
- `check_uniqueness_enforcement`: LiveCompare がユーザー定義を使用している場合比較キー、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から、LiveCompare は、設定されている場合、比較キーでテーブルの一意性をチェックします `check_uniqueness_enforcement` が有効になっています (デフォルト)。
- `oracle_ignore_unsortable`: 有効にすると、列を無視するよう LiveCompare に指示します列がそうでない場合、オラクルのソート不可能なデータ型 (`BLOB`、`CLOB`、`NCLOB`、`BFILE`) テーブル PK のパート。この設定を有効にする場合は、有効にすることもお勧めします `column_intersection`。
- `oracle_user_tables_only`: 有効にすると、テーブルをフェッチするよう LiveCompare に指示しますオラクルのログインユーザからのみのメタデータ。これは、読み取り、例、`sys.user_tables` およ

び `sys.user_tab_columns` からではなく `sys.all_tables` および `sys.all_tab_columns`。デフォルト：`off`。

- `oracle_fetch_fk_metadata`：有効にすると、LiveCompare に外部をフェッチするように指示しますキーメタデータ（遅いオペレーションであるかもしれません）。設定の値をオーバーライドしますオラクル接続の `fetch_fk_metadata`。デフォルト：`off`。
- `schema_qualified_table_names`：テーブル名はスキーマ修飾として扱われますこの設定が有効になっている場合。無効にすると、テーブルを比較せずに比較できますスキーマ修飾テーブル名の使用：オラクル x Postgres の比較では、`oracle_user_tables_only` も有効にする必要がありますが、Postgres x Postgres では、異なるスキーマの下にあるテーブルの比較を可能にします。同じデータベース。また、`schema_qualified_table_names` が有効な場合、Table Filter -> tables、Row Filter、および Column Filter はテーブル名前を許可しますスキーマ名前なし。デフォルト：`on`。
- `force_collate`：`off` 以外の値と有効な照合順序に設定されている場合名前、すべての ORDER BY 操作で指定された照合順序名前を強制します比較される Postgres データベース。Postgres データベースと比較するときに役立ちます異なる照合順序、またはオラクルデータベースと Postgres データベースを比較する場合（これでケースのユーザーは `force_collate = C` を設定する必要があります。比較する場合、値 C を想定します混在テクノロジー（オラクルと PostgreSQL など）、照合順序は指定されていません。デフォルト：`off`。
- `work_directory`：LiveCompare 作業ディレクトリへのパス。セッション出力ファイルを含むフォルダは、そのようなディレクトリに作成されます。デフォルト：`.`（カレントディレクトリ）。
- `abort_on_setup_error`：有効な場合、LiveCompare がエラーをヒットした場合テーブル比較ラウンドを設定しようとする、比較セッション全体が中止しました。デフォルト：`off`。

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

- `custom_dollar_quoting_delimiter`：LiveCompare が違いを見つけると、文字列にドル引用符付け符を使用して DML を出力します。デフォルトの動作は作成ですそれを構成するランダム文字列。何らかの方法でカスタムのもを使用したい場合は、このパラメータを使用するデリミタとして設定できます。あなただけを設定する必要があります定数は、定数を囲む\$シンボルではありません。デフォルトは `off` で、これは LiveCompare は、LiveCompare ワードの 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 は使用します `session_replication_role` 出力の `replica` としての PostgreSQL 設定スクリプト。トリガーとルールの起動を防ぎたい場合に便利です分岐のあるノードに DML を適用します。有効にするには PostgreSQL が必要ですそれ以外の場合、スーパーユーザは無効になります。デフォルトは `off` です。
- `split_updates`：LiveCompare を有効にすると、UPDATE ダイバージェンスが分割されます。つまり、UPDATE DML を生成する代わりに、対応する適用スクリプトの DELETE および INSERT。デフォルトは `off` です。
- `float_point_round`：LiveCompare の 10 進数数を指定する整数データベースからの浮動小数点値を比較するときに丸める必要があります。デフォルト
- 1 で、浮動小数点の丸めを無効にします。

2.8.2 初期接続

初期接続は、`logical_replication_mode` が `pglogical` または `bdr` に設定されている場合にのみ使用され、ユーザが `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、ノード名から DSN をフェッチするためにプログラムの起動時にのみ使用されます。

- `technology` : RDBMS テクノロジー。現在、唯一可能な値は `postgresql`。
- `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、他のすべてを設定することもできます `parameter key words allowed by libpq`。
- `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` : データベース名前。 デフォルト : `postgres`。
- `user` : データベースユーザ。 デフォルト : `postgres`。
- `application_name`。 アプリケーション名前。 ユーザが `dsn` を設定した場合でも使用できます他のすべての接続情報の代わりに。 デフォルト : `livecompare_initial`。

2.8.3 出力接続

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

- `technology` : RDBMS テクノロジー。現在、唯一可能な値は `postgresql`。
- `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、他のすべてを設定することもできます `parameter key words allowed by libpq`。
- `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` : データベース名前。 デフォルト : `postgres`。
- `user` : データベースユーザ。 デフォルト : `postgres`。
- `application_name`。 アプリケーション名前。 ユーザが `dsn` を設定した場合でも使用できます他のすべての接続情報の代わりに。 デフォルト : `livecompare_output`。

2.8.4 データ接続

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

Initial Connection 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。セクション名前は、Name Connection のフォームである必要があります。Name は、アルファベット文字で始まる単一単語の文字列です。この場合、ユーザが Name に入力するものはすべて、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作成するため、ユーザはデータ接続を指定する必要はありません。

- **technology** : RDBMS テクノロジー。現在可能な値は postgresql または oracle。
- **node_name** : クラスター内のノードの名前。が必要です logical_replication_mode は pglogical または bdr に設定されており、以下も必要です。Initial Connection がいっぱいになります。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、他のすべてを設定することもできます parameter key words allowed by libpq
- **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** : データベース名前。デフォルト : postgres。
- **service** : オラクル接続で使用されるサービス名前。デフォルト : XE。
- **user** : データベースユーザ。デフォルト : postgres。
- **password** : プレーンテキストパスワード。これを使用しないことをお勧めしますが、レガシー接続が必要になる場合があります。
- **application_name** . アプリケーション名前。ユーザが dsn を設定した場合でも使用できますまたは、他のすべての接続情報の代わりに node_name。デフォルト : livecompare_<Connection ID>。
- **start_query** : への接続のたびに実行される任意のクエリーデータベースが開いています。
- **fetch_fk_metadata** : LiveCompare が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接続で。デフォルト : on。

2.8.5 テーブルフィルター

省略するか、空のままにすると、.ini ファイルのこのセクションは、LiveCompare が **first** データベースの **all** テーブルに対して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 を実行する場合は、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 はそ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。変数 **publication_name** を使用して条件式を作成できます。例：

```
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、特定のレプリケーションセット、および LiveCompare は、関連付けられているテーブルでのみ機能します。それらのレプリケーションセット。変数 **set_name** を使用して、条件式、例：

```
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ます。LiveCompare はそれらのスキーマに属するテーブル。変数 **schema_name** を使用できます条件式を作成するには、例：

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

- **tables** : 変数 **table_name** は、条件の作成にヘルプます LiveCompare が動作するテーブルのみをフィルタリング式 例：

```
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ノートしてください。

schema_qualified_table_names が無効になっていない限り、テーブル名前はスキーマで修飾する必要があります。例、特定のテーブルのリストのみをフィルタリングすることができます。

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

一般設定 **schema_qualified_table_names** を無効にした場合、接続 **start_query** 設定で Postgres に適切な **search_path** も設定する必要があります。次に例を示します。

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

[My Connection]
...
```

(次のページに続く)

(前のページからの続き)

```
start_query = SET search_path TO myschema1, myschema2
```

```
[Table Filter]
```

```
tables = table_name in (mytable1, mytable2)
```

重要：ノートに設定された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、最初に見つかったスキーマのみ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`、`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積部分にあるテーブルのセットを考慮します。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、テーブルフィルタが最初のデータベースに適用されることにノートしてください。テーブルが最初のデータベースに存在し、フィルタで考慮されているが、他のデータベースには存在しない場合、このようなログがログに表示され、その特定のテーブルの比較はスキップされたされます。

```
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くても、比較では考慮されません。

特定のテーブルの比較は、テーブル・カラム名が完全に同じでない場合（`column_intersection` が有効になっていない限り）、同じオーダーでもスキップされたされます。適切なメッセージもログファイルに含まれます。

現在、LiveCompare は両方のテーブルでデータ型もコンストレインも同じ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を使用しない `make` にノートしてください。

2.8.6 行フィルター

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

このセクションの下に、テーブルのリストを 1 行に 1 つのテーブルで記述できます (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で修飾する必要が例ます)。

[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。

一般設定 schema_qualified_table_names を無効にした場合、接続 start_query 設定で Postgres に適切な search_path も設定する必要があります。次に例を示します。

[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
```

あらゆる種類の SQL 条件 (WHERE 句に入れるのと同じ) が、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。例、ラージテーブルがあり、特定の数の ID のみを比較する場合、すべての ID を含む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)
```

行フィルターの記述が正しくない場合、LiveCompare はフィルターの適用を試みますが、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たされ、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 によってフィルターで除外された場合、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使用しない make にノートしてください。

2.8.7 列フィルター

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

このセクションの下に、行ごとに1つのテーブルのリストを作成できます(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で修飾する必要があります)。例、public.table1 と public.table2 の両方に column1、column2、column3、column4 および column5 の列がある とします。

[Column Filter]

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

この場合、テーブル public.table1 では、LiveCompare は列 column2、column4 および column5 でのみ機能し、column1 および column3 を除外します。一方、テーブル public.table2 では、列 column2、フィルタリングおよび column3 およびフィルタリング column1 および column5 から。

一般設定 schema_qualified_table_names を無効にした場合、接続 start_query 設定で Postgres に適切な search_path も設定する必要があります。次に例を示します。

[General Setting]

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

[My Connection]

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

[Column Filter]

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

列フィルターに列名が指定されていない場合、つまり列が指定されたテーブルに存在しない場合、LiveCompare は有効な列のみを使用して、検出できなかった列に関するメッセージをログに記録し、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、どういうわけか Table Filter によってフィルターで除外された場合、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 のパートである場合、比較では無視されないことにノートしてください。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使用しない make にノートしてください。

2.8.8 比較キー

Column Filter と同様に、このセクションでは、テーブルごとに列のリストを指定することもできます。テーブルに主キーまたは UNIQUE 制約がある場合でも、これらの列は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` の場合、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、テーブルの除外または欠落に関する同じ動作は、Comparison Key にも適用されます。同様に、競合モードでは Comparison Key セクションは無視されます。

2.8.9 競合フィルター

このセクションでは、BDR ノードから競合をフェッチするときに、`--conflicts` モードで使用されるフィルターを指定できます。任意の SQL 条件式を作成し、式の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前
- `relname`：リレーションする関係のリレーション名前

セクションの下で `conflicts` 属性を使用する必要があります。以下の例をご覧ください。

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

上記の設定を INI ファイルに追加することにより、LiveCompare は、`update_missing` 型の競合だけをフェッチし、各 BDR ノードの競合を照会しながらスキーマ `my_schema` の下のテーブルに関連します。

重要：このセクションはノートモード専用です。

2.9 Licenses

2.9.1 TQDM

`tqdm` は、共同作業の成果物です。特に明記されていない限り、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 ライセンスの下で作品をリリースします（以下のテキスト）。

以下に、例外または著名な著者を新しいオーダーに示します。

- ファイル： *MPLv2.0 2015-2020 (c) キャスパーダコスタルイス casperdcl。
- ファイル：tqdm / _tqdm.py Google Inc. を代表する MIT 2016 (c) [PR # 96]
- ファイル：tqdm / _tqdm.py setup.py README.rst MANIFEST.in .gitignore MIT 2013 (c) Noam Yorav-Raphael、オリジナル著者。

[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には、Mozilla Public ライセンス v. 2.0 の条件がサブジェクトされます。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。

2.9.2 MIT ライセンス (MIT)

Copyright (c) 2013 noamraph

これにより、このソフトウェアおよび関連する文書ファイル（「ソフトウェア」）のコピーを取得するすべての人に、使用、コピー、変更、マージの権利を含むがこれらに制約ないソフトウェアをフリーに処理する許可が付与されますソフトウェアのコピーをパブリッシュ、配布、サブライセンス、および/または販売し、以下のサブジェクトに従って、ソフトウェアの提供先にソフトウェアの提供を許可します。

上記の著作権表示およびこのパーミッション通知は、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、商品性、特定の目的への適合性、および権利の非侵害の保証を含むがこれに限定されない、明示または黙示を問わず、いかなる保証もなしに「現状のまま」提供されます。いかなる場合においても、作者または著作権者は、契約、不法行為、またはその他の行為、ソフトウェアまたは使用またはその他の取引から生じた、または出た、またはこれらに関連するいかなる請求、損害またはその他の責任についても責任を負わないものとしますソフトウェア。

2.9.3 cx_Oracle

CX_ORACLE のライセンス契約

Copyright © 2016、2020、オラクルおよび/またはその関連会社。禁無断転載

Copyright © 2007-2015、Anthony Tuininga。禁無断転載

Copyright © 2001-2007、Computronix (Canada) Ltd.、エドモントン、アルバータ、カナダ。禁無断転載

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

ソースコードの再配布では、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。バイナリフォームでの再配布では、上記の著作権表示、この条件リスト、および以下の免責事項を、ディストリビューション物とともに提供される文書および/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、書面による事前のパーミッションなしに、このソフトウェアから派生した製品を推奨またはプロモートに使用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者および寄稿者によって**現状のまま**提供され、商品性および特定の目的への適合性の黙示的な保証を含むがこれに限定されない明示的または黙示的な保証は放棄されます。いかなる場合でも、再生者または貢献者は、直接的、間接的、偶発的、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むが、これらに限定されない）について責任を負わないものとします業務中断）いかなる理由であれ、契約、厳格責任、または不法行為（過失またはその他を含む）のいずれの場合でも、責任の理論上、このソフトウェアの使用から生じる可能性があります。

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

© Copyright 2016、2020、オラクルおよび/またはその関連会社。禁無断転載部分 Copyright © 2007-2015、Anthony Tuininga。禁無断転載一部の Copyright © 2001-2007、Computronix (Canada) Ltd.、エドモントン、アルバータ、カナダ。すべての権利は改訂版 10e5c258 に予約。

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、このドキュメントのセクション 1~9 で定義されている使用、複製、およびディストリビューションの条件を意味するものとします。

「ライセンサー」とは、著作権所有者、また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エンティティを意味します。

「法的エンティティ」とは、活動するエンティティと、そのエンティティを制御する、制御される、またはそのエンティティと共通の制御下にある他のすべてのエンティティのユニオンを意味するものとします。この定義の目的のために、「制御」とは、(i) 直接または間接を問わず、契約またはその他の方法で、そのようなエンティティの方向または管理を引き起こす力、または (ii) 50 % (50 %) の所有権、またはより多くの発行済み株式、または (iii) そのようなエンティティの受益権。

(次のページに続く)

(前のページからの続き)

「あなた」(または「あなた」)とは、このライセンスによって付与される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、文書ソース、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、生成された文書、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から生じるフォームを意味するものとします。

「著作物」とは、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例に、ソース形式またはオブジェクトフォームにかかわらず、ライセンスに基づいて提供される著作物の著作物を意味するものとします。

「派生著作物」とは、ソース形式またはオブジェクトフォームにかかわらず、著作物に基づく（または派生する）編集改訂、注釈、推 **bor**、またはその他の修正が全体としてオリジナルの著作物を表す著作物を意味するものとします。本ライセンスの目的のために、派生著作物は、その著作物および派生著作物から分離可能なままである、または単にその作品および派生著作物のインターフェースにリンク（または名前でバインド）する著作物を含みません。

「貢献」とは、著作物の所有者または個人によって著作権物に **inclusion** ために意図的にライセンサーに提出された著作物のオリジナルバージョン、およびその著作物またはその派生著作物への修正または追加を含む著作者の著作物を意味するものとします。著作権所有者に代わって提出する権限を持つ法人。この定義の目的のために、「提出」とは、電子メーリングリスト、ソースコード管理システム、および問題追跡システムでの通信を含むがこれらに限定されない、ライセンサーまたはその代表者に送信されるあらゆるフォームの電子的、口頭、または書面による通信を意味します。著作物の議論および改善を目的として、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「寄付ではない」と顕著にマークされているか書面で指定されているコミュニケーションは除きます。

「貢献者」とは、ライセンサーと、ライセンサーが貢献を受け取り、その後作品に組み込まれた個人または法人を意味するものとします。

2. Copyright 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、派生的著作権の複製、準備、公開、公演、サブライセンス、および著作物およびそのような派生著作物をソースまたはオブジェクトフォームで配布します。

3. 特許ライセンスの付与。このライセンスの条件に従い、各コントリビューターは、お客様に、永続的、世界的、非独占的、無料、ロイヤルティフリー、取消不能（このセクションに記載されている場合を除く）特許ライセンスを付与 **make**、作品の使用、販売、販売、インポート、その他の譲渡。このようなライセンスは、貢献者のみまたは貢献者との組み合わせによって必ず侵害される貢献者によってライセンス可能な特許請求にのみ適用されます。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が直接または寄与特許侵害であると主張するエンティティ（訴訟における相互請求または反訴を含む）に対して特許訴訟を起こした場合、そのためにこのライセンスの下で付与された特許ライセンス訴訟は、そのような訴訟が提出された日付をもって終了するものとします。

4. 再配布。次の条件を満たす場合は、改変の有無にかかわらず、ソースまたはオブジェクトフォームで、作品またはその派生作品のコピーを任意の媒体で複製および配布できます。

(a) 作品または派生作品の他の受領者には、このライセンスのコピーを提供する必要があります。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示す顕著な通知を表示させる必要があります。そして

(c) 配布する派生物のソースフォームで、著作権、特許、商標、および著作物のソースフォームからのすべての帰属通知を、派生物のいかなるパートにも関係しない通知を除き、保持する必要があります作品；そして

(d) 作品の配布のパートとして「通知」テキストファイルが含まれている場合、ディストリビューションする派生作品には、そのような通知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。次の場所の少なくとも 1 つにある、**Derivative Works** のパート：**Derivative Works** のパートとして配布された **NOTICE** テキストファイル内。派生著作物と一緒に提供される場合、ソースフォームまたは文書内。または、**Derivative Works** によって生成されたディスプレイ内で、そのようなサードパーティの通知が通常表示される場合はいつでも。**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。そのような追加の帰属通知がライセンスの変更と解釈できない場合、著作物の通知テキストに加えて、または追加として、配布する派生著作物内に独自の帰属通知を追加できます。

修正に独自の著作権を追加することができ、修正の使用、複製、またはディストリビューションのための追加または異なるライセンス条件を提供することができディストリビューション。それ以外の場合、作品は本ライセンスに記載されている条件に適合します。

5. 寄付の提出。明示的に別段の定めがない限り、お客様が著作物に inclusion ためにライセンサーに意図的に提出した貢献物は、追加の条件なしで、このライセンスの条件に従います。上記にかかわらず、本契約のいかなる条項も、かかる貢献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することも、変更することはありません。

6. 商標。このライセンスは、作品のオリジンを説明し、**NOTICE** ファイルのコンテンツを複製する際の合理的かつ慣例的な使用に必要な場合を除き、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パーミッションを与えません。

7. 保証の否認。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、明示または黙示を問わず、明示または黙示を問わず、「現状有姿」で作品を提供します。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。お客様は、作品の使用または再配布の適切性を判断する責任を単独で負い、本ライセンスに基づく許可の練習に関連するリスクを負います。

8. 責任の制限。不法行為（過失を含む）、契約、その他のいかなる場合においても、適用法（意図的および重大な過失行為など）で要求されない限り、または書面で合意した場合を除き、いかなるイベントも法理的理論の下で、本ライセンスの結果として、または作品の使用または使用不能から生じたキャラクターの直接的、間接的、特別、偶発的、または結果的損害を含む損害（のれんの損失、仕事の中断を含むがこれらに限定されない）、コンピュータの故障または誤動作、またはその他のあらゆる商業的損害または損失）、そのようなコントリビューターにそのような損害の可能性が通知されている場合でも。

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する際、お客様は、サポート、保証、補償、またはこのライセンスと一致するその他の責任義務および/または権利の提供を受け入れ、料金を請

求することができます。ただし、かかる義務を承諾する場合、お客様は、他のコントリビューターに代わってではなく、お客様自身の責任およびお客様の単独の責任でのみ行動することができます。またはあなたが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貢献者に対して主張された主張。

利用規約の終了

2.9.4 Psycopg2

psycopg2 はフリーソフトウェアです。FreeSoftware Foundation が発行する GNU Lesser General Public ライセンスの条件、ライセンスのバージョン 3、または（オプションで）それ以降のバージョンのいずれかで、再配布および/または変更できます。

psycopg2 は有用であることを期待して配布されていますが、いかなる保証也没有ありません。商品性または特定の目的への適合性の暗黙の保証さえありません。詳細については、GNU Lesser General Public ライセンスを参照してください。

さらに、特別な例外として、著作権所有者は、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 の修正バージョン）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パーミッションを与えます。

OpenSSL 以外で使用するすべてのコードについて、あらゆる点で GNU Lesser General Public ライセンス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、変更する義務はありません。そうしたくない場合は、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、ここでも削除します。

psycopg2 とともに GNU Lesser General Public ライセンス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。

代替ライセンス-----~~~~~

次の BSD ライクライセンスは、パターン `psycopg/adapter*. {h,c}` および `psycopg/microprotocol*. {h,c}` に続くファイルに（オプションで）適用されます。

次のサブジェクトに従って、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を許可します。

1. このソフトウェアのオリジンを不実表示してはなりません。オリジナルのソフトウェアを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は、製品の文書に謝辞を入れていただければ幸いです、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伝えてはなりません。
3. この通知は、ソースディストリビューション物から削除または変更することはできません。

2.9.5 表

Copyright (c) 2011-2020 Sergey Astanin and contributors

これにより、このソフトウェアおよび関連する文書ファイル（「ソフトウェア」）のコピーを取得するすべての人に、使用、コピー、変更、マージの権利を含むがこれらに制約ないソフトウェアをフリーに処理する許可が付与されますソフトウェアのコピーをパブリッシュ、配布、サブライセンス、および/または販売し、以下のサブジェクトに従って、ソフトウェアの提供先にソフトウェアの提供を許可します。

上記の著作権表示およびこのパーミッション通知は、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、商品性、特定の目的への適合性、および権利の非侵害の保証を含むがこれに限定されない、明示または黙示を問わず、いかなる保証もなしに「現状のまま」提供されます。いかなる場合においても、作者または著作権者は、契約、不法行為、またはその他の行為、ソフトウェアまたは使用またはその他の取引から生じた、または出た、またはこれらに関連するいかなる請求、損害またはその他の責任についても責任を負わないものとしますソフトウェア。© 2020 GitHub, Inc.

2.9.6 OmniDB

MIT ライセンス

部分 Copyright (c) 2015-2019、OmniDB チーム部分 Copyright (c) 2017-2019、2ndQuadrant Limited

これにより、このソフトウェアおよび関連する文書ファイル（「ソフトウェア」）のコピーを取得するすべての人に、使用、コピー、変更、マージの権利を含むがこれらに制約ないソフトウェアをフリーに処理する許可が付与されますソフトウェアのコピーをパブリッシュ、配布、サブライセンス、および/または販売し、以下のサブジェクトに従って、ソフトウェアの提供先にソフトウェアの提供を許可します。

上記の著作権表示およびこのパーミッション通知は、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、商品性、特定の目的への適合性、および権利の非侵害の保証を含むがこれに限定されない、明示または黙示を問わず、いかなる保証もなしに「現状のまま」提供されます。いかなる場合においても、作者または著作権者は、契約、不法行為、またはその他の行為、ソフトウェアまたは使用またはその他の取引から生じた、または出た、またはこれらに関連するいかなる請求、損害またはその他の責任についても責任を負わないものとしますソフトウェア。