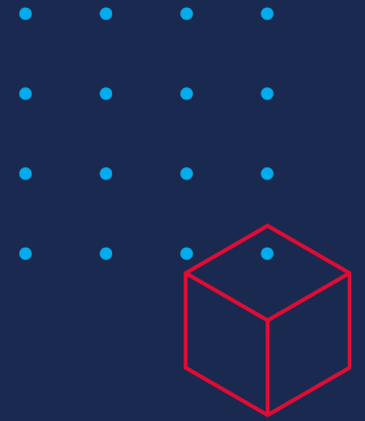

開発者向けの TiDB キット

PingCAP Training and Certification による TiDB Cloud と TiUP Playground のミニデモ

免責事項

- この資料のすべてのサンプルコードは、デモと学習のみを目的としています
- このリポジトリのサンプルコードは、評価なしで本番環境で使用しないでください

道場の準備:TiDB Cloud または TiUP Playground

- **Step 1:**: Github からデモリポジトリをクローンします
  - **git clone https://github.com/pingcap/tidb-course-201-lab.git**
- **Step 2a:** TiDB Cloud Serverless を選択します
  - ***TiDB Cloud キックスタートワークショップの演習 1 に従って Serverless Tier Cluster を作成します***
- **Step 2b:** **または** ローカルマシンで TiUP Playground を実行することを選択してください (Linux または macOS の場合)

```
// Step 2a
$ export TIDB_CLOUD_HOST={hostname}
$ export TIDB_CLOUD_USERNAME={username}
$ export TIDB_CLOUD_PASSWORD={password}
$ export TIDB_CLOUD_PORT={port}
```

```
// Step 2b
$ curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/install.sh | sh
$ git clone https://github.com/pingcap/tidb-course-201-lab.git
$ cd tidb-course-201-lab/scripts && ./playground-start.sh
```

ミニデモマニフェスト

- **B:** ベスト・プラクティス
- **K:** ナレッジ

B1: JDBC バッチインサート	B2: Python バッチインサート		
K1: 一般的なデータ型の最大長	K2: 文字セット (UTF8MB4 と GBK)	K3: AUTO_INCREMENT	K4: AUTO_RANDOM
K5: Clustered と Non-Clustered PK	K6: オプティミスティック TX ロック	K7: 悲観的 TX ロック	K8: フィードデータベース Kafka へのイベント変更
K9: RAW KV サンプル (beta)	K10: オンラインスキーマの変更	K11: JDBC TLS 接続	

B1: JDBC バッチインサート

- 環境:Java SDK
- スクリプトとサンプルコード:
  - ***10-demo-jdbc-batch-insert-01-show.sh***
  - ***DemoJdbcBatchInsert.java***
- ミニデモストーリー:
  - スクリプトを実行すると、JDBC パラメーター **rewriteBatchedStatements=true** を使用して **10000** 行をテストテーブルに挿入します
  - すると、スクリプトは **rewriteBatchedStatements** を **false** に設定した状態で再度実行します
  - 両方の実行の経過時間の違いを確認します
- TiDB Cloud でデモを実行の場合は 「cloud」、ローカルの Playground でデモを実行する場合は 「local」 をそれぞれ引数に指定します

```
// 1. Go to the working directory: tidb-course-201-lab/scripts
$ cd tidb-course-201-lab/scripts

// 2. Run the demo script
$ ./10-demo-jdbc-batch-insert-01-show.sh cloud|local
```

B1: JDBC バッチインサート (出力)

- 次のサンプル出力は、ローカルの TiUP Playground から生成されました
- クライアントプログラムと TiDB Cloud が同じ Region に含まれていない場合、2 回の実行間の elapsed time のギャップはかなり大きくなります
  - **rewriteBatchedStatements=false** の実行が完了するのを待ちきれない場合は、 **ctrl-c** を押して終了します

```
$ ./10-demo-jdbc-batch-insert-01-show.sh local
TiDB Endpoint:127.0.0.1
TiDB Username:root
Connection established.
>>> Begin insert 10000 rows.
>>> End batch insert,rewriteBatchedStatements=true,elapsed: 285 (ms).
Connection established.
>>> Begin insert 10000 rows.
>>> End batch insert,rewriteBatchedStatements=false,elapsed: 11226 (ms).

/* Executing query: select count(*), max(name) from test.t1_batchtest; */
        Row#, count(*), max(name)
        1) 10000, 9999
Turn on autocommit.
Connection closed.
```

B2: Python バッチインサート

- 環境:Python 3.9
- スクリプトとサンプルコード:
  - *10-demo-python-batch-insert-01-show.sh*
  - *demo-batch-insert.py*
- ミニデモストーリー:
  - バッチスタイルのステートメント **INSERT INTO ... VALUES (),(),(),...** を使用して **10000** 行を 1 つのテーブルに挿入するスクリプトを実行します
  - スクリプトはループスタイルを再度使用し、一度に 1 行ずつ挿入します
  - 両方の実行の経過時間の違いを確認します
- TiDB Cloud でデモを実行の場合は 「cloud」、ローカルの Playground でデモを実行する場合は 「local」 をそれぞれ引数に指定します

```
// 1. Go to working directory: tidb-course-201-lab/scripts
$ cd tidb-course-201-lab/scripts

// 2. Run demo script
$ ./10-demo-python-batch-insert-01-show.sh cloud|local
```

B2: Python バッチインサート (出力)

- 次のサンプル出力は、ローカルの TiUP Playground から生成されました
- クライアントプログラムと TiDB Cloud が同じリージョンにない場合、2 つの実行間の elapsed time のギャップはかなり大きくなります
  - 2 回目の非バッチ形式の実行が完了するのを待ちきれない場合は、 **ctrl-c** を押して終了します

```
$ ./10-demo-python-batch-insert-01-show.sh local
...
Connected to TiDB: root@127.0.0.1:4000
Batch Inserting 10000 rows in 104.645751953125 (ms).
Total rows in t1_batchtest table: 10000.
Non-Batch Inserting 10000 rows in 5803.891845703125 (ms).
Total rows in t1_batchtest table: 10000.
```

K1: 一般的なデータ型の最大長

- 環境:Python 3.9
- スクリプトとサンプルコード:
  - *03-demo-data-type-maxlength-01-show.sh*
  - *demo-data-type-maxlength.py*
- ミニデモストーリー:
  - 文字セットは **utf8mb4** と仮定します
  - 最大長のデータを含む行をサンプルテーブルに挿入し、結果を表示する
  - **TIMESTAMP** データ型のクエリ値はタイムゾーンによって異なります
  - 以下のデータ型の最大サイズは、以下の設定の組み合わせによって制約されます
    - TiDB サーバー: **txn-entry-size-limit (default: 6291456 bytes)** と **txn-total-size-limit (default: 104857600 bytes)**
    - TiKV サーバー: **raft-entry-max-size (default: 8 MB)**
    - **MEDIUMTEXT**、 **LONGTEXT**、 **MEDIUMBLOB**、 **LONGBLOB**、 **JSON**

```
// 1. Go to working directory: tidb-course-201-lab/scripts
$ cd tidb-course-201-lab/scripts

// 2. Run demo script
$ ./03-demo-data-type-maxlength-01-show.sh cloud|local
```

一般的なデータ型の K1: の最大長 (出力)

- 次のサンプル出力は TiDB Serverless Tier から生成されました

```
$ ./03-demo-data-type-maxlength-01-show.sh cloud
Connected to TiDB: 2v▧▧▧▧▧▧▧▧▧▧▧▧7K.root@▧▧▧▧▧▧▧▧▧▧.us-west-2.prod.aws.tidbcloud.com:4▧▧▧
...
BINARY(255): 255 Bytes
CHAR(255): 1020 Bytes [255 Chars]
VARCHAR(16383): 65532 Bytes [16383 Chars]
TINYTEXT: 255 Bytes
TEXT: 65535 Bytes
MEDIUMTEXT: 6291405 Bytes + a few Bytes
LONGTEXT: 6291407 Bytes + a few Bytes
TINYBLOB: 255 Bytes
BLOB: 65535 Bytes
MEDIUMBLOB: 6291405 Bytes + a few Bytes
LONGBLOB: 6291407 Bytes + a few Bytes
JSON: 6291391 Bytes + a few Bytes
YEAR_MIN: 0
YEAR_MAX: 2155
DATE_MIN: 0001-01-01
DATE_MAX: 9999-12-31
TIME_MIN: -34 days, 15:59:59.999999
TIME_MAX: 34 days, 15:59:59.999999
DATETIME_MIN: 0001-01-01 00:00:01
DATETIME_MAX: 9999-12-31 23:59:59.999999
TIMESTAMP_MIN: 1970-01-01 08:00:01
TIMESTAMP_MAX: 2038-01-19 11:14:07.999999
...
```

K2: キャラクタセット (UTF8MB4 と GBK)

- 環境:mysql-client
- スクリプトとサンプルコード:
  - *03-demo-charset-01-show.sql*
- ミニデモストーリー:
  - **フィックス 3** 機能を使用して **UTF8MB4** と **GBK** をテストします
  - **フィックス 3** と **GBK** では、一般的な **CJK** 文字が 1 文字に対して **3** バイトを消費することがわかります

```
// 1. Go to working directory: tidb-course-201-lab/scripts
$ cd tidb-course-201-lab/scripts

// 2. Connect to TiDB

      // Connect to TiDB Cloud
      $ ./connect-cloud.sh

      // Connect to local Playground
      $ ./connect-4000.sh

// 3. Call the demo script
tidb:4000> source 03-demo-charset-01-show.sql

// 4. Close the connection with TiDB or TiDB Cloud
tidb:4000> exit;
```

K2: キャラクタセット (UTF8MB4 と GBK) (出力)

- 次のサンプル出力は、ローカルの TiUP Playground から生成されました

```
$ ./connect-4000.sh
tidb:4000> source 03-demo-charset-01-show.sql
*************************** 1. row ***************************
    Byte_Length: 5
    Char_Length: 5
        English: Hello
    GBK_ENCODED: Hello
UTF8MB4_ENCODED: Hello
     GBK_BINARY: 0x48656C6C6F
 UTF8MB4_BINARY: 0x48656C6C6F
1 row in set (0.00 sec)

*************************** 1. row ***************************
    Byte_Length: 15
    Char_Length: 5
       Japanese: こんにちは
    GBK_ENCODED: こんにちは
UTF8MB4_ENCODED: こんにちは
     GBK_BINARY: 0xA4B3A4F3A4CBA4C1A4CF
 UTF8MB4_BINARY: 0xE38193E38293E381ABE381A1E381AF
1 row in set (0.00 sec)

*************************** 1. row ***************************
    Byte_Length: 6
    Char_Length: 2
        Chinese: 你好
    GBK_ENCODED: 你好
UTF8MB4_ENCODED: 你好
     GBK_BINARY: 0xC4E3BAC3
 UTF8MB4_BINARY: 0xE4BDA0E5A5BD
1 row in set (0.00 sec)
```

K3: AUTO_INCREMENT

- 環境:mysql-client、TiUP Playground
- スクリプトとサンプルコード:
  - ***07-demo-auto-increment-01-setup.sql***
  - ***07-demo-auto-increment-03-show.sh***
- ミニデモストーリー:
  - このデモは TiUP Playground **のみ** です
  - **AUTO_INCREMENT** と **AUTO_ID_CACHE 300** を使用してテーブルを作成する
  - **2** TiDB サーバーインスタンスから新しい行を挿入し、結果を確認します

```
// 1. Go to working directory: tidb-course-201-lab/scripts
$ cd tidb-course-201-lab/scripts

// 2. Run demo scripts
$ ./07-demo-auto-increment-01-setup.sh
$ ./07-demo-auto-increment-03-show.sh
```

K3: AUTO_INCREMENT (出力)

- 次のサンプル出力は、ローカルの TiUP Playground から生成されました

```
$ ./07-demo-auto-increment-01-setup.sh
$ ./07-demo-auto-increment-03-show.sh
INSERT via TiDB server 4000
INSERT via TiDB server 4001
id      from_port
1       4000
2       4000
301     4001
302     4001
```

K4: AUTO_RANDOM

- 環境:mysql-client
- スクリプトとサンプルコード:
  - *07-demo-auto-random-01-show.sql*
- ミニデモストーリー:
  - **AUTO_RANDOM(4)** 属性を使用してテーブルを作成し、数行挿入して結果を確認する
  - 最後のクエリは n 行を返すはずですが、n は 16 である 2^4 に近いのはなぜですか？

```
// 1. Go to working directory: tidb-course-201-lab/scripts
$ cd tidb-course-201-lab/scripts

// 2. Connect to TiDB

      // Connect to TiDB Cloud
      $ ./connect-cloud.sh

      // Connect to local Playground
      $ ./connect-4000.sh

// 3. Call the demo script
tidb:4000> source 07-demo-auto-random-01-show.sql

// 4. Close the connection with TiDB or TiDB Cloud
tidb:4000> exit;
```

K4: AUTO_RANDOM (出力)

- 次のサンプル出力は、ローカルの TiUP Playground から生成されました

```
./connect-4000.sh
tidb:4000> source 07-demo-auto-random-01-show.sql
...

+----------------------------+--------------+
| TIDB_ROW_ID_SHARDING_INFO | TIDB_PK_TYPE |
+----------------------------+--------------+
| PK_AUTO_RANDOM_BITS=4      | CLUSTERED    |
+----------------------------+--------------+
1 row in set (0.01 sec)

+-----------+----------------------+
| id_prefix | approx_rows_in_shard |
+-----------+----------------------+
| 11        |                   11 |
| 17        |                    6 |
| 23        |                    5 |
| 28        |                    7 |
| 34        |                    9 |
| 40        |                    5 |
| 46        |                    8 |
| 51        |                    6 |
| 57        |                   15 |
| 63        |                   10 |
| 69        |                    6 |
| 74        |                    5 |
| 80        |                    3 |
| 86        |                    6 |
+-----------+----------------------+
14 rows in set (0.00 sec)
```

K5: JDBC TLS コネクション

- 環境:Java SDK
- スクリプトとサンプルコード:
  - ***tls.toml***
  - ***playground-start-with-tls.sh***
  - ***12-demo-jdbc-connection-secured-01-show.sh***
  - ***DemoJdbcConnectionSecured.java***
- ミニデモストーリー:
  - このデモは TiUP Playground ***のみ*** です
  - **auto-tls** を有効にした状態で Playground を作成する
  - 複数の **sslMode** 設定を使用して TiDB サーバーインスタンスに接続し、違いを確認します

```
// 1. Stop the default Playground in terminal 1
$ <ctrl-c>

// 2. Start a TLS enabled Playground in terminal 1
$ ./playground-start-with-tls.sh

// 3. In another terminal, Go to working directory: tidb-course-201-lab/scripts
$ cd tidb-course-201-lab/scripts
```

```
// 4. Run demo script
$ ./12-demo-jdbc-connection-secured-01-show.sh local

// 5. Stop the TLS enabled Playground by pressing ctrl-c, wait until the command prompt returns
$ <ctr-c>

// 6. Clean up the environment and restart the default Playground in terminal 1
$ ./playground-clean-classroom-tls.sh
$ ./playground-start.sh
```

K5: JDBC TLS コネクション (アウトプット)

- 次のサンプル出力は TiDB Cloud Serverless Tier から生成されました

```
$ ./12-demo-jdbc-connection-secured-01-show.sh cloud
TiDB endpoint: ▧▧▧▧▧▧▧▧▧.us-west-2.prod.aws.tidbcloud.com
TiDB username: 2v▧▧▧▧▧▧▧▧▧▧▧▧7K.root
Default TiDB server port: 4▧▧▧
...
### Trying with sslMode=DISABLED ###
Error: java.sql.SQLNonTransientConnectionException: Connections using insecure transport are prohibited.
...
### Trying with sslMode=REQUIRED ###
Connection established.
...
        1) Ssl_cipher, TLS_AES_128_GCM_SHA256
...
### Trying with sslMode=PREFERRED ###
Connection established.
...
        1) Ssl_cipher, TLS_AES_128_GCM_SHA256
...
### Trying with sslMode=VERIFY_CA ###
Connection established.
...
        1) Ssl_cipher, TLS_AES_128_GCM_SHA256
...
### Trying with sslMode=VERIFY_IDENTITY ###
Connection established.
...
        1) Ssl_cipher, TLS_AES_128_GCM_SHA256
...
```

K6: Clustered と Non-Clustered のプライマリキー

- 環境:mysql-client
- スクリプトとサンプルコード:
  - ***07-demo-compare-clustered-and-nonclustered-pk.sql***
- ミニデモストーリー:
  - **Clustered PK** を使用してテーブル 1 を作成します
  - **Non-Clustered PK** を使用してテーブル 2を作成し、テーブル 1からデータをコピーします
  - どちらのテーブルにも同じデータがあり、約 200 万行です
  - PK の範囲述語について、TiKV Regions カウントと物理実行計画を比較します
    - 実行計画から、 **Non-Clustered PK** の場合はもう 1つ操作が必要であることがわかります

```
// 1. Go to working directory: tidb-course-201-lab/scripts
$ cd tidb-course-201-lab/scripts

// 2. Connect to TiDB

      // Connect to TiDB Cloud
      $ ./connect-cloud.sh

      // Connect to local Playground
      $ ./connect-4000.sh

// 3. Call the demo script
tidb> source 07-demo-compare-clustered-and-nonclustered-pk.sql
```

K6: Clustered および Non-Clustered プライマリキー (出力)

- 次のサンプル出力は TiDB Cloud Serverless Tier から生成されました

```
$ ./connect-cloud.sh
tidb> source 07-demo-compare-clustered-and-nonclustered-pk.sql
...
+-----------------------------+---------------------------------+
| Clustered # of TiKV Regions | Non-Clustered # of TiKV Regions |
+-----------------------------+---------------------------------+
|                           3 |                               6 |
+-----------------------------+---------------------------------+
1 row in set (0.02 sec)

+-------------------------------------------------------------------------------+
| Clustered                                                                     |
+-------------------------------------------------------------------------------+
| SELECT varname FROM test.auto_increment_t2_clustered WHERE id between 10 and 100; |
+-------------------------------------------------------------------------------+

+--------------------------+---------+-----------+-----------------------------------+---------------------------------------+
| id                       | estRows | task      | access object                     | operator info                         |
+--------------------------+---------+-----------+-----------------------------------+---------------------------------------+
| Projection_4             | 88.93   | root      |                                   | test.auto_increment_t2_clustered.varname |
| └─TableReader_6          | 88.93   | root      |                                   | data:TableRangeScan_5                 |
|   └─TableRangeScan_5     | 88.93   | cop[tikv] | table:auto_increment_t2_clustered | range:[10,100], keep order:false      |
+--------------------------+---------+-----------+-----------------------------------+---------------------------------------+

+-----------------------------------------------------------------------+
| Non-Clustered                                                         |
+-----------------------------------------------------------------------+
| SELECT varname FROM test.bigint_t3_nonclustered WHERE id between 10 and 100; |
+-----------------------------------------------------------------------+

+---------------------------------+---------+-----------+-----------------------------------------------+-----------------------------------+
| id                              | estRows | task      | access object                                 | operator info                     |
+---------------------------------+---------+-----------+-----------------------------------------------+-----------------------------------+
| Projection_4                    | 92.93   | root      |                                               | test.bigint_t3_nonclustered.varname |
| └─IndexLookUp_10                | 92.93   | root      |                                               |                                   |
|   ├─IndexRangeScan_8(Build)     | 92.93   | cop[tikv] | table:bigint_t3_nonclustered, index:PRIMARY(id) | range:[10,100], keep order:false  |
|   └─TableRowIDScan_9(Probe)     | 92.93   | cop[tikv] | table:bigint_t3_nonclustered                  | keep order:false                  |
+---------------------------------+---------+-----------+-----------------------------------------------+-----------------------------------+
4 rows in set (0.00 sec)
```

K7: オプティミスティック・トランザクション・ロック

- 環境:Java SDK
- スクリプトとサンプルコード:
  - ***09-demo-jdbc-tx-optimistic-01-show.sh***
  - ***DemoJdbcTxOptimisticLock.java***
- ミニデモストーリー:
  - **optimistic mode** では、2 つのトランザクションが同じ行を同時に更新すると競合が発生することがある
  - このスクリプトには **no-retry** と **retry** の 2 つのオプションがあり、実行すると異なる結果を確認できます
    - **no-retry:** トランザクションは **ErrorCode 9007** の前に自動的にロールバックされます
    - **retry:** セッションで **ErrorCode 9007** が発生した場合は、失敗した DML を再試行します

```
// 1. Go to the working directory: tidb-course-201-lab/scripts
$ cd tidb-course-201-lab/scripts

// 2. Call the demo script twice with no-retry and retry options
$ ./09-demo-jdbc-tx-optimistic-01-show.sh cloud|local no-retry
$ ./09-demo-jdbc-tx-optimistic-01-show.sh cloud|local retry
```

K7: オプティミスティック・トランザクション・ロック (no-retry オプションの出力)

- 次のサンプル出力は TiDB Cloud Serverless Tier から生成されました

```
$ ./09-demo-jdbc-tx-optimistic-01-show.sh cloud no-retry
TiDB endpoint: ▧▧▧▧▧▧▧▧▧.us-west-2.prod.aws.tidbcloud.com
TiDB username: 2v▧▧▧▧▧▧▧▧▧▧▧▧7K.root
Default TiDB server port: 4▧▧▧
Security options: &sslMode=VERIFY_IDENTITY&enabledTLSProtocols=TLSv1.3
Connection established.
Connection A session started
Connection B session started
Connection A session: BEGIN OPTIMISTIC
Connection A session: UPDATE test_tx_optimistic SET name = 'Connection A' WHERE id = 864691128455135233
Connection B session: BEGIN OPTIMISTIC
Connection B session: UPDATE test_tx_optimistic SET name = 'Connection B' WHERE id = 864691128455135233
Connection B session: Commit
Connection B session: Checking result

/* Executing query: select id, name from test_tx_optimistic; */
        Row#, id, name
        1) 864691128455135233, Connection B

Connection A session: Commit
Connection A ErrorCode: 9007
Connection A SQLState: HY000
Connection A Error: java.sql.SQLException: Write conflict, txnStartTS=434395274207297539,
conflictStartTS=434395274469441537, conflictCommitTS=434395274993729538,
key={tableID=5836, handle=864691128455135233} primary={tableID=5836, handle=864691128455135233} [try again later]
< Session in Connection A RAISED THE EXCEPTION !!! >
Connection A session: Checking result

/* Executing query: select id, name from test_tx_optimistic; */
        Row#, id, name
        1) 864691128455135233, Connection B
```

K7: オプティミスティック・トランザクション・ロック (retry オプションの出力)

- 次のサンプル出力は TiDB Cloud Serverless Tier から生成されました

```
$ ./09-demo-jdbc-tx-optimistic-01-show.sh cloud retry
TiDB endpoint: ▧▧▧▧▧▧▧▧▧.us-west-2.prod.aws.tidbcloud.com
TiDB username: 2v▧▧▧▧▧▧▧▧▧▧▧▧7K.root
Default TiDB server port: 4▧▧▧
Security options: &sslMode=VERIFY_IDENTITY&enabledTLSProtocols=TLSv1.3
Connection established.
Connection B session started
Connection A session started
Connection A session: BEGIN OPTIMISTIC
Connection A session: UPDATE test_tx_optimistic SET name = 'Connection A' WHERE id = 5188146770730811393
Connection B session: BEGIN OPTIMISTIC
Connection B session: UPDATE test_tx_optimistic SET name = 'Connection B' WHERE id = 5188146770730811393
Connection B session: Commit
Connection B session: Checking result

/* Executing query: select id, name from test_tx_optimistic; */
        Row#, id, name
        1) 5188146770730811393, Connection B

Connection A session: Commit
Connection A ErrorCode: 9007
Connection A SQLState: HY000
Connection A Error: java.sql.SQLException: Write conflict, txnStartTS=434395314905415681,
conflictStartTS=434395315167559681, conflictCommitTS=434395315691847682,
key={tableID=5839, handle=5188146770730811393} primary={tableID=5839, handle=5188146770730811393} [try again later]
< Session in Connection A RAISED THE EXCEPTION !!! >
Connection A session: Commit
Connection A session: Checking result

/* Executing query: select id, name from test_tx_optimistic; */
        Row#, id, name
        1) 5188146770730811393, Connection A
```

K8: ペシミスティック・トランザクション・ロック

- 環境:Java SDK
- スクリプトとサンプルコード:
  - ***09-demo-jdbc-tx-pessimistic-01-show.sh***
  - ***DemoJdbcPessimisticLock.java***
- ミニデモストーリー:
  - **pessimistic mode** では、2 つのトランザクションが同じ行を同時に更新しても競合は発生しません
  - ブロックされたセッションは、トランザクションロックが解除されるのを待ちます
  - **Errorcode 9007** はありません

```
// 1. Go to working directory: tidb-course-201-lab/scripts
$ cd tidb-course-201-lab/scripts

// 2. Run demo script
$ ./09-demo-jdbc-tx-pessimistic-01-show.sh cloud|local
```

K8: ペシミスティック・トランザクション・ロック (出力)

- 次のサンプル出力は TiDB Cloud Serverless Tier から生成されました

```
$ ./09-demo-jdbc-tx-pessimistic-01-show.sh cloud
TiDB endpoint: ■■■■■■■■■■.us-west-2.prod.aws.tidbcloud.com
TiDB username: 2v■■■■■■■■■■■■■7K.root
Default TiDB server port: 4■■■■
Security options: &sslMode=VERIFY_IDENTITY&enabledTLSProtocols=TLSv1.3
Connection established.
Connection B session started
Connection A session started
Connection A session: BEGIN PESSIMISTIC
Connection A session: UPDATE test_tx_optimistic SET name = 'Connection A' WHERE id = 1729382256910270465
Connection B session: BEGIN PESSIMISTIC
Connection B session: UPDATE test_tx_optimistic SET name = 'Connection B' WHERE id = 1729382256910270465
Connection A session: Commit
Connection A session: Checking result

/* Executing query: select id, name from test_tx_optimistic; */
        Row#, id, name
        1) 1729382256910270465, Connection A

Connection B session: Commit
Connection B session: Checking result

/* Executing query: select id, name from test_tx_optimistic; */
        Row#, id, name
        1) 1729382256910270465, Connection B
```

K9: は TiCDC を使用してデータベースの変更を Kafka にフィードします

- 環境:TiUP Playground、Kafka、mysql-client
- 準備:
  - ***Kafka* ディストリビューションのダウンロード**
- ミニデモストーリー:
  - このデモは TiUP Playground 専用です
  - ローカルの Kafka サービスとコンシューマーを起動する
  - open-protocol (他のプロトコルも利用可能) を使用して TiCDC キャプチャ changefeed タスクを作成する
  - DDL、DML を希望どおりに実行し、Kafka コンシューマー側からキャプチャされた変更イベントを確認します

K9: は TiCDC を使用してデータベースの変更を Kafka にフィードします (デモステップ)

```
// 1. Stop the default Playground you started previously on terminal 1
$ <ctrl-c>

// 2. Start Zookeeper: On terminal 1 - under the folder you downloaded the Kafka TAR ball, e.g: version 2.13-3.2.0
$ tar -xzf kafka_2.13-3.2.0.tgz
$ cd kafka_2.13-3.2.0
$ bin/zookeeper-server-start.sh config/zookeeper.properties

// 3. Start Kafka Service: On terminal 2 - under the folder you installed the Kafka binary
$ bin/kafka-server-start.sh config/server.properties

// 4. Create a Kafka Topic: On terminal 3 - under the folder you installed the Kafka binary
$ bin/kafka-topics.sh --create --topic cdc-example-topic --bootstrap-server localhost:9092

// 5. Start Kafka Console Consumer: On terminal 3 - under the folder you installed the Kafka binary
$ bin/kafka-console-consumer.sh --topic cdc-example-topic --from-beginning --bootstrap-server localhost:9092

// 6. Start Playground: On terminal 4
$ tiup playground v6.1.1 --tag cdc-example --db 2 --pd 3 --kv 3 --ticdc 1 --tiflash 1

// 7. Create a TiCDC Change Feed Task: terminal 5
$ cd tidb-course-201-lab/scripts
$ ./13-demo-cdc-create-changefeed-01.sh

// 8. Do Any Changes by Executing DDL/DML in terminal 5, and OBSERVE the captured changes on terminal 3
$ ./connect-4000.sh
tidb:4000> create table test.t10 (id bigint primary key);
tidb:4000> insert into test.t10 values (100);
tidb:4000> ...

// 9. Clean up the environment
        // Tear Down: On terminal 4, 3, 2, 1
        $ Press <ctrl-c> in terminal 4, 3, 2, 1 in order
        $ tiup clean cdc-example

// 10. Restart the default Playground on terminal 1
$ ./playground-start.sh
```

K9: は TiCDC 経由でデータベースの変更を Kafka にフィードします (サンプル出力)

- ターミナル 5 は TiDB サーバーに接続されています
- ターミナル 3 は Kafka Topic Consumer に接続されています

```
// On terminal 5, execute CREATE/INSERT/UPDATE/DELETE in order
tidb:4000> create table test.t10 (id bigint primary key);
Query OK, 0 rows affected (0.25 sec)

tidb:4000> insert into test.t10 values (100);
Query OK, 1 row affected (0.01 sec)

tidb:4000> update test.t10 set id=200 where id=100;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

tidb:4000> delete from test.t10;
Query OK, 1 row affected (0.02 sec)

// On terminal 3, you can see four events for DDL, INSERT, UPDATE and finally the DELETE
$ bin/kafka-console-consumer.sh --topic cdc-example-topic --from-beginning --bootstrap-server localhost:9092

A{"q":"CREATE TABLE `test`.`t10` (`id` BIGINT PRIMARY KEY)","t":3}
,{"u":{"id":{"t":8,"h":true,"f":11,"v":100}}}
,{"d":{"id":{"t":8,"h":true,"f":11,"v":100}}},{"u":{"id":{"t":8,"h":true,"f":11,"v":200}}}
,{"d":{"id":{"t":8,"h":true,"f":11,"v":200}}}
```

K10: 未処理の KV サンプル

- 環境:Python 3.9
- スクリプトとサンプルコード:
  - ***01-demo-simple-raw-kv.sh***
  - ***demo-simple-put-get-rawkv.py***
- ミニデモストーリー:
  - このデモは **TiUP Playground *のみ***
  - 実験的段階の Python API を使用して TiKV ストアとして TiKV にアクセスする

```
// 1. Go to working directory: tidb-course-201-lab/scripts
$ cd tidb-course-201-lab/scripts

// 2. Run demo script
$ ./01-demo-simple-raw-kv.sh
```

K10: 未処理の KV サンプル (出力)

- 次のサンプル出力は、ローカルの TiUP Playground から生成されました

```
$ ./01-demo-simple-raw-kv.sh
...
put(b'Key1',b'Value1')
Jul 14 09:54:26.997 INFO connect to tikv endpoint: "127.0.0.1:20161"
get(b'Key1'): b'Value1'
get(b'Key0'): None
```

K11: オンライン Schema チェンジ

- 環境:Java SDK、mysql-client
- スクリプトとサンプルコード:
  - ***11-demo-jdbc-prepared-statement-online-ddl-01-show.sh***
  - ***DemoJdbcPreparedStatement8028.java***
  - ***07-demo-online-ddl-add-column-02.sql***
- ミニデモストーリー:
  - セッション **A** はワークロードを実行して行 (合計 **192000** 行) を挿入します
    - ワークロードスクリプト: **11-demo-jdbc-prepared-statement-online-ddl-01-show.sh**
  - セッション **B** は、DDL を実行して、セッション **フィックス 3** が行を挿入しているテーブルに新しいカラムを追加します
    - 注:TiDB では DML は DDL をブロックしません。その逆も同様です
  - 最初のデモ実行では、エラーコードヒントなしでワークロードを実行して、オンライン DDL がセッション **フィックス 3** の DML にどのように影響するかを確認します
  - 2 回目のデモ実行では、エラーコード **8028** が発生したときにトランザクションを再実行するようにプログラムに指示するために、オプション **8028** を 2番目のパラメータとしてワークロードを実行します
    - **Error code 8028**: Information schema is changed during the execution of the statement
- パラメータ **[cloud|local]** を使用して、それぞれ TiDB Cloud またはローカル Playground に対してデモを実行します
- デモの実行手順の詳細は、次のスライドに記載されています...

K11: オンライン Schema チェンジ

- デモステップ

```
// 1. Go to working directory: tidb-course-201-lab/scripts
$ cd tidb-course-201-lab/scripts

// 2. FIRST demo run - ErrorCode: 8028 is NOT handled

      // On terminal 1, call script to run the inserting workload without error handling hint
      $ ./11-demo-jdbc-prepared-statement-online-ddl-01-show.sh cloud|local

      // When terminal 1 begin to inserting rows, in terminal 2, connect to TiDB with mysql-client
          // Connect to TiDB Cloud
          $ ./connect-cloud.sh

          // Or, connect to local Playground
          $ ./connect-4000.sh

      // On terminal 2, call script to trigger an Online DDL on the workload table
      tidb:4000> source 07-demo-online-ddl-add-column-02.sql

      // Observe what happened in terminal 1, is workload interrupted? How many rows had been inserted?

// 3. SECOND demo run - ErrorCode: 8028 is handled for once

      // On terminal 1, call script to run the inserting workload with error handling hint this time
      $ ./11-demo-jdbc-prepared-statement-online-ddl-01-show.sh cloud|local 8028

      // When terminal 1 begin to inserting rows, in terminal 2, connect to TiDB with mysql-client
          // Connect to TiDB Cloud
          $ ./connect-cloud.sh

          // Or, connect to local Playground
          $ ./connect-4000.sh

      // On terminal 2, call script to trigger an Online DDL on the workload table
      tidb:4000> source 07-demo-online-ddl-add-column-02.sql

      // Observe what happened in terminal 1, is workload interrupted? How many rows had been inserted?
```

K11: 初回デモ実行 (アウトプット)

- 次のサンプル出力は、ローカルの TiUP Playground から生成されました

```
// On terminal 1
 $ ./11-demo-jdbc-prepared-statement-online-ddl-01-show.sh local
TiDB endpoint: 127.0.0.1
TiDB username: root
Default TiDB server port: 4000
Connection established.
preparing
...
populating
...
Error: java.sql.SQLException: Information schema is changed during the execution of the statement
(for example, table definition may be updated by other DDL ran in parallel). If you see this error often,
try increasing `tidb_max_delta_schema_count`. [try again later]
SQLState: HY000
ErrorCode: 8028
...
Connection closed.
```

K11: 2 回目のデモ実行 (アウトプット)

- 次のサンプル出力は、ローカルの TiUP Playground から生成されました

```
// On terminal 1
 $ ./11-demo-jdbc-prepared-statement-online-ddl-01-show.sh local 8028
TiDB endpoint: 127.0.0.1
TiDB username: root
Default TiDB server port: 4000
Connection established.
preparing
...
populating
...
Error: java.sql.SQLException: Information schema is changed during the execution of the statement
(for example, table definition may be updated by other DDL ran in parallel). If you see this error often,
try increasing `tidb_max_delta_schema_count`. [try again later]
SQLState: HY000
ErrorCode: 8028
...
8028 (schema mutation) encountered, backoff...
DO anything in reaction to error, in this example we continue our workload.
...
        Row#, name1, |BEFORE-DDL-GOAL: 192000|
        1) BEFORE-DDL, 192000
I: 199
Turn on autocommit.
Connection closed.
```

Learn TiDB from PingCAP

- *TiDB ドキュメント*
- *TiDB Cloud ドキュメント*
- *PingCAP トレーニングおよび認定ポータル*
- *TiDB Cloud キックスタートワークショップ*

Thanks.