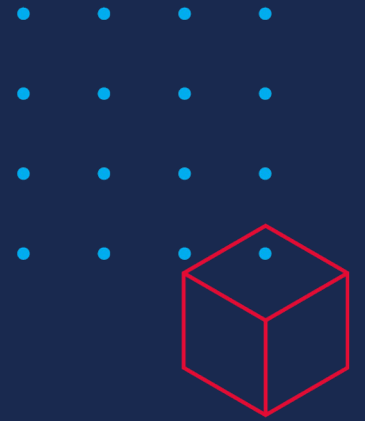

TiDB Kits for Developers

Mini demos on TiDB Cloud and TiUP Sandbox by PingCAP Training and Certification (20221201)

Disclaimer

- All sample codes in this material are for demo and study purpose only.
- Do NOT use the sample codes in this repository in production without evaluation.

Prepare the Dojo: TiDB Cloud or TiUP Playground

- **Step 1:** Clone the demo repository from Github
  - **`git clone https://github.com/pingcap/tidb-course-201-lab.git`**
- **Step 2a:** Choose TiDB Cloud Serverless
  - ***Create Serverless Tier cluster by following the exercise 1 in TiDB Cloud Kickstart Workshop***
- **Step 2b:** **Or** choose running TiUP Playground on local machine (for Linux or macOS)

```
// 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
```

Mini Demo Manifest

- **B:** Best practices
- **K:** TiDB Knowledges

B1: JDBC Batch Insert	B2: Python Batch Insert		
K1: Maximum Length for Common Data Types	K2: Character Set (UTF8MB4 and GBK)	K3: AUTO_INCREMENT	K4: AUTO_RANDOM
K5: Clustered and Non-Clustered PK	K6: Optimistic TX Lock	K7: Pessimistic TX Lock	K8: Feed Database Change Events to Kafka
K9: Raw KV Example (beta)	K10: Online Schema Change	K11: JDBC TLS Connection	

B1: JDBC Batch Insert

- Environment: Java SDK
- Scripts and sample codes:
  - *10-demo-jdbc-batch-insert-01-show.sh*
  - *DemoJdbcBatchInsert.java*
- Mini demo story:
  - Run the script, and it will insert **10000** rows into one table with JDBC parameter **rewriteBatchedStatements** set to **true**
  - Then, the script will do it again with **rewriteBatchedStatements** set to **false**
  - Observe the differences on the elapsed time for both runs
- Use parameter **cloud|local** to run the demo against TiDB Cloud or local Playground respectively

```
// 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 Batch Insert (Output)

- Following sample output was generated from local TiUP Playground
- If the client program and TiDB Cloud are not in the same Region, the elapsed time gap between two runs will be quite large
  - In case you cannot wait for the **`rewriteBatchedStatements=false`** run to complete, feel free to press **`ctrl-c`** to quit

```
$ ./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 Batch Insert

- Environment: Python 3.9
- Scripts and sample codes:
  - *10-demo-python-batch-insert-01-show.sh*
  - *demo-batch-insert.py*
- Mini demo story:
  - Run the script to insert **10000** rows into one table using batch style statement **INSERT INTO ... VALUES (),(),(),...**
  - The script will do it again using a loop style again, and inserting one row at a time
  - Observe the differences on the elapsed time for both runs
- Use parameter **cloud|local** to run the demo against TiDB Cloud or local Playground respectively

```
// 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 Batch Insert (Output)

- Following sample output was generated from local TiUP Playground
- If the client program and TiDB Cloud are not in the same region, the elapsed time gap between two executions will be quite large
  - In case you cannot wait for the second non-batch style run to complete, feel free to press **`ctrl-c`** to quit

```
$ ./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: Maximum Length for Common Data Types

- Environment: Python 3.9
- Scripts and sample codes:
  - *03-demo-data-type-maxlength-01-show.sh*
  - *demo-data-type-maxlength.py*
- Mini demo story:
  - Assume the character set is **utf8mb4**
  - Insert rows with the maximum length data into example tables, then show the result
  - **TIMESTAMP** data type queried value is timezone dependent
  - The following data type maximum sizes are constrained by following settings combined:
    - TiDB: **txn-entry-size-limit (default: 6291456 bytes)** and **txn-total-size-limit (default: 104857600 bytes)**
    - TiKV: **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
```

K1: Maximum Length for Common Data Types (Output)

- Following sample output was generated from TiDB Serverless Tier

```
$ ./03-demo-data-type-maxlength-01-show.sh cloud
Connected to TiDB: 2v[redacted]7K.root@[redacted].us-west-2.prod.aws.tidbcloud.com:4[redacted]
...
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: Character Set (UTF8MB4 and GBK)

- Environment: mysql-client
- Scripts and sample codes:
  - *03-demo-charset-01-show.sql*
- Mini demo story:
  - Test **UTF8MB4** and **GBK** with **CAST** function
  - You can tell that common **CJK** characters consume **3** bytes for single character in **UTF8MB4** and **GBK**

```
// 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: Character Set (UTF8MB4 and GBK) (Output)

- Following sample output was generated from local TiUP Playground

```
$ ./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

- Environment: mysql-client, TiUP Playground
- Scripts and sample codes:
  - *07-demo-auto-increment-01-setup.sql*
  - *07-demo-auto-increment-03-show.sh*
- Mini demo story:
  - This demo is TiUP Playground **ONLY**
  - Create a table with **AUTO_INCREMENT** and **AUTO_ID_CACHE 300**
  - Insert new rows from **two** TiDB server instances, observe the result

```
// 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 (Output)

- Following sample output was generated from local TiUP Playground

```
$ ./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

- Environment: mysql-client
- Scripts and sample codes:
  - *07-demo-auto-random-01-show.sql*
- Mini demo story:
  - Create a table with **AUTO_RANDOM(4)** attribute, insert a few rows, then check the result
  - The last query should return n rows and n is close to 2^4 which is 16, why?

```
// 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 (Output)

- Following sample output was generated from local TiUP Playground

```
./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 Connection

- Environment: Java SDK
- Scripts and sample codes:
  - ***tls.toml***
  - ***playground-start-with-tls.sh***
  - ***12-demo-jdbc-connection-secured-01-show.sh***
  - ***DemoJdbcConnectionSecured.java***
- Mini demo story:
  - This demo is TiUP Playground **`ONLY`**
  - Create a Playground with **`auto-tls`** enabled
  - Connect to TiDB server instance with several **`sslMode`** settings and observe the difference

```
// 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 Connection (Output)

- Following sample output was generated from TiDB Cloud Serverless Tier

```
$ ./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 and Non-Clustered Primary Key

- Environment: mysql-client
- Scripts and sample codes:
  - ***07-demo-compare-clustered-and-nonclustered-pk.sql***
- Mini demo story:
  - Create table 1 with **`Clustered PK`**
  - Create table 2 with **`Non-Clustered PK`**, copy data from table 1
  - Both tables have the same data which is around 2 million rows
  - Compare their TiKV Regions count and physical execution plans on range predicates on PK
    - From the execution plan, you will find that **`Non-Clustered PK`** case takes one more operation

```
// 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 and Non-Clustered Primary Key (Output)

- Following sample output was generated from TiDB Cloud Serverless Tier

```
$ ./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: Optimistic Transaction Lock

- Environment: Java SDK
- Scripts and sample codes:
  - *09-demo-jdbc-tx-optimistic-01-show.sh*
  - *DemoJdbcTxOptimisticLock.java*
- Mini demo story:
  - In **`optimistic mode`**, two transactions update the same row at the same time might cause conflict
  - The script provides two options: **`no-retry`** and **`retry`**, run it to observe the different results:
    - **`no-retry:`** Transaction will be automatically rolled back in front of **`ErrorCode 9007`**
    - **`retry:`** Retry the failed DML if your session encounter **`ErrorCode 9007`**

```
// 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: Optimistic Transaction Lock (Output for no-retry Option)

- Following sample output was generated from TiDB Cloud Serverless Tier

```
$ ./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: Optimistic Transaction Lock (Output for retry Option)

- Following sample output was generated from TiDB Cloud Serverless Tier

```
$ ./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: Pessimistic Transaction Lock

- Environment: Java SDK
- Scripts and sample codes:
  - *09-demo-jdbc-tx-pessimistic-01-show.sh*
  - *DemoJdbcPessimisticLock.java*
- Mini demo story:
  - In **`pessimistic mode`**, two transactions update the same row at the same time cannot cause conflict
  - The blocked session will wait for the transaction lock to be released
  - No **`Errorcode 9007`**

```
// 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: Pessimistic Transaction Lock (Output)

- Following sample output was generated from TiDB Cloud Serverless Tier

```
$ ./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: Feeds Database Changes to Kafka by Using TiCDC

- Environment: TiUP Playground, Kafka, mysql-client
- Preparation:
  - ***Download Kafka distribution***
- Mini demo story:
  - This demo is TiUP Playground ONLY
  - Start the local Kafka service and consumer
  - Create a TiCDC capture changefeed task using open-protocol (other protocols available)
  - Execute DDL, DML as you wish, and observe the captured change events from Kafka consumer side

K9: Feeds Database Changes to Kafka by Using TiCDC (Demo Steps)

```
// 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: Feeds Database Changes to Kafka Via TiCDC (Sample Output)

- Terminal 5 is connected to TiDB server
- Terminal 3 is attached to Kafka Topic Consumer

```
// 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: Raw KV Example

- Environment: Python 3.9
- Scripts and sample codes:
  - ***01-demo-simple-raw-kv.sh***
  - ***demo-simple-put-get-rawkv.py***
- Mini demo story:
  - This demo is **`TiUP Playground ONLY`**
  - Accesss TiKV as a KV store via experimental Python API

```
// 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: Raw KV Example (Output)

- Following sample output was generated from local TiUP Playground

```
$ ./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: Online Schema Change

- Environment: Java SDK, mysql-client
- Scripts and sample codes:
  - ***11-demo-jdbc-prepared-statement-online-ddl-01-show.sh***
  - ***DemoJdbcPreparedStatement8028.java***
  - ***07-demo-online-ddl-add-column-02.sql***
- Mini demo story:
  - Session **A** runs a workload to insert rows, **`192000`** rows in total.
    - Workload script: **`11-demo-jdbc-prepared-statement-online-ddl-01-show.sh`**
  - Session **B** executes a DDL to add a new column for the table which session **A** is inserting rows into.
    - Note: DML does not block DDL in TiDB, and vice versa
  - In the 1st demo run, execute the workload without the error code hint to observe the result that how online DDL affects the DML for session **A**.
  - In the 2nd demo run, execute the workload with option **`8028`** as the second parameter to tell the program to re-execute the transaction when encountering error code **`8028`**.
    - **`Error code 8028`**: Information schema is changed during the execution of the statement.
- Use parameter **`[cloud|local]`** to run the demo against TiDB Cloud or local Playground respectively.
- Detail demo run steps are listed in the next slide ...

K11: Online Schema Change

- Demo steps

```
// 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: First Demo Run (Output)

- Following sample output was generated from local TiUP Playground

```
// 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: Second Demo Run (Output)

- Following sample output was generated from local TiUP Playground

```
// 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 Document***
- ***TiDB Cloud Document***
- ***PingCAP Training & Certification Portal***
- ***TiDB Cloud Kickstart Workshop***

Thanks.