TiDB Ninja Kits

PingCAP Training and Certification (2022071201) - Mini demos on TiDB Cloud and TiUP Sandbox

Disclaimer

- All sample codes in this material are for Demo/Study purpose only
- Do NOT use them in production without evaluation

Choose Dojo: TiDB Cloud or TiUP Playground

- **TiDB Cloud**
  - Go to **https://tidbcloud.com**, and create a FREE developer tier TiDB cluster in the Cloud
  - Create a TiDB Cloud Cluster
  - Set environment variables for TiDB Cloud:
    - `$ export TIDB_HOST=<TiDB Cloud Cluster DNS Name>`
    - `$ export TIDB_USERNAME=<Database Username>`
    - `$ export TIDB_PASSWORD=<User Password>`
- **TiUP Playground** on Linux or macOS
  - `$ curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/install.sh | sh`
  - Add **tiup** command to PATH: `$ source ~/.bash_profile`
  - Set environment variables for local Playground sandbox:
    - `$ export TIDB_HOST=127.0.0.1`
    - `$ export TIDB_USERNAME=root`

Mini Demo Manifest

Title	Title	Title	Title
B1: JDBC Batch Insert	K1: Maximum Length for Common Data Types	K2: Characterset (UTF8MB4 and GBK)	KP3: AUTO_INCREMENT
K4: AUTO_RANDOM	K5: Java Client TLS Connection	K6: Clustered and Non-Clustered PK	K7: Optimistic TX Lock
K8: Pessimistic TX Lock	KP9: Feed DB Changes to Kafka via TiCDC		

B1: JDBC Batch Insert

- Environment: **`Java SDK`**
- Sample code:
  - Line 42, 64, 66: DemoJdbcBatchInsert.java
- Mini demo story:
  - Run script to insert **`10000`** rows into one table with **`rewriteBatchedStatements`** set to **`true`**
  - Then, the script will do it again with **`rewriteBatchedStatements`** set to **`false`**
  - Obverse the differences on elapsed times

```
// Setup: Terminal 1 - Skip this step if you are testing on TiDB Cloud
$ tiup playground v6.1.0 --tag batch-example --db 2 --pd 3 --kv 3 --tiflash 1

// Demo Run: Terminal 2
$ git clone https://github.com/pingcap/tidb-course-201-lab.git
$ cd tidb-course-201-lab/scripts
$ ./10-demo-jdbc-batch-insert-01-show.sh

// Tear Down: Terminal 1 - Skip this step if you are testing on TiDB Cloud
$ <ctrl-c>
$ tiup clean batch-example
```

B1: JDBC Batch Insert (Output)

- Following output example is from TiUP playground
- If the client 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 **`rewriteBatchedStatements=false`** run to complete, feel free to hit **`ctrl-c`**

```
$ ./10-demo-jdbc-batch-insert-01-show.sh
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.
```

K1: Maximum Length for Common Data Types

- Environment: **`Python 3.x`**
- Sample code:
  - 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 types are constrained by **`txn-entry-size-limit`** and **`raft-entry-max-size`** settings combined:
    - **`MEDIUMTEXT`**, **`LONGTEXT`**, **`MEDIUMBLOB`**, **`LONGBLOB`**, **`JSON`**

```
// Setup: Terminal 1 - Skip this step if you are testing on TiDB Cloud
$ tiup playground v6.1.0 --tag maxlen-example --db 2 --pd 3 --kv 3 --tiflash 1

// Demo Run: Terminal 2
$ git clone https://github.com/pingcap/tidb-course-201-lab.git
$ cd tidb-course-201-lab/scripts
$ ./03-demo-data-type-maxlength-01-show.sh

// Tear Down: Terminal 1 - Skip this step if you are testing on TiDB Cloud
$ <ctrl-c>
$ tiup clean maxlen-example
```

K1: Maximum Length for Common Data Types (Output)

```
$ ./03-demo-data-type-maxlength-01-show.sh
Connected to TiDB: root@127.0.0.1:4000
...
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: Characterset (UTF8MB4 and GBK)

- Environment: **`mysql-client`**
- Sample code:
  - 03-demo-charset-01-show.sql
- Mini demo story:
  - Test **`UTF8MB4`** and **`GBK`** with **`CAST`** function

```
// Setup: Terminal 1 - Skip this step if you are testing on TiDB Cloud
$ tiup playground v6.1.0 --tag cs-example --db 2 --pd 3 --kv 3 --tiflash 1

// Demo Run: Terminal 2
$ git clone https://github.com/pingcap/tidb-course-201-lab.git
$ cd tidb-course-201-lab/scripts
$ ./connect-4000.sh
tidb> source 03-demo-charset-01-show.sql

// Tear Down: Terminal 1 - Skip this step if you are testing on TiDB Cloud
$ <ctrl-c>
$ tiup clean cs-example
```

K2: Characterset (UTF8MB4 and GBK) (Output)

```
tidb> 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`**
- Sample code:
  - Line 9, 14, 16: 07-demo-auto-increment-01-setup.sql
  - Line 6, 8: 07-demo-auto-increment-03-show.sh
- Mini demo story:
  - **`TiUP Playground`** only demo
  - Create a table with **`AUTO_INCREMENT`** and **`AUTO_ID_CACHE 300`**
  - Insert new rows from **`2`** TiDB-Server instances, observe the result

```
// Setup: Terminal 1
$ tiup playground v6.1.0 --tag ai-example --db 2 --pd 3 --kv 3 --tiflash 1

// Demo Run: Terminal 2
$ git clone https://github.com/pingcap/tidb-course-201-lab.git
$ cd tidb-course-201-lab/scripts
$ ./07-demo-auto-increment-01-setup.sh
$ ./07-demo-auto-increment-03-show.sh

// Tear Down: Terminal 1
$ <ctrl-c>
$ tiup clean ai-example
```

K3: AUTO_INCREMENT (Output)

```
$ ./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`**
- Sample code:
  - Line 11, 137~141: 07-demo-auto-random-01-show.sql
- Mini demo story:
  - Create a table with **`AUTO_RANDOM(4)`** attribute, insert some rows, then check the result
  - The last query should return **`n`** rows and **`n`** is close to **`2^4`** which is **`16`**, why?

```
// Setup: Terminal 1 - Skip this step if you are testing on TiDB Cloud
$ tiup playground v6.1.0 --tag ar-example --db 2 --pd 3 --kv 3 --tiflash 1

// Demo Run: Terminal 2
$ git clone https://github.com/pingcap/tidb-course-201-lab.git
$ cd tidb-course-201-lab/scripts
$ ./connect-4000.sh
tidb> source 07-demo-auto-random-01-show.sql

// Tear Down: Terminal 1 - Skip this step if you are testing on TiDB Cloud
$ <ctrl-c>
$ tiup clean ar-example
```

K4: AUTO_RANDOM (Output)

```
tidb> 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: Java TLS Connection without Server and Client Verification

- Environment: **`Java SDK`**
- Sample code:
  - Sample TiDB for auto-tls: tls.toml
  - Line 34, 40, 42: DemoJdbcConnectionSecured.java
- Mini demo story:
  - Create a playground with **`auto-tls`** enabled (skip in TiDB Cloud case)
  - Connect to TiDB server instance with several **`sslMode`** settings and observe the difference

```
// Setup: Terminal 1 - Skip this step if you are testing on TiDB Cloud
$ git clone https://github.com/pingcap/tidb-course-201-lab.git
$ cd tidb-course-201-lab/scripts
$ ./playground-start-with-tls.sh

// Demo Run: Terminal 2
$ cd tidb-course-201-lab/scripts
$ ./12-demo-jdbc-connection-secured-01-show.sh

// Tear Down: Terminal 1 - Skip this step if you are testing on TiDB Cloud
$ <ctrl-c>
$ cd tidb-course-201-lab/scripts
$ ./playground-clean-classroom-tls.sh
```

K5: Java TLS Connection without Server and Client Verification (Output)

```
$ ./12-demo-jdbc-connection-secured-01-show.sh
...
### Trying with sslMode=DISABLED ###
...
        1) Ssl_cipher,
...
### Trying with sslMode=REQUIRED ###
...
        1) Ssl_cipher, TLS_AES_128_GCM_SHA256
...
### Trying with sslMode=PREFERRED ###
...
        1) Ssl_cipher, TLS_AES_128_GCM_SHA256
...
### Trying with sslMode=VERIFY_CA ###
Error: com.mysql.cj.jdbc.exceptions.CommunicationsException: Communications link failure
...
### Trying with sslMode=VERIFY_IDENTITY ###
Error: com.mysql.cj.jdbc.exceptions.CommunicationsException: Communications link failure
...
```

K6: Clustered and Non-Clustered Primary Key

- Environment: **`mysql-client`**
- Sample code:
  - Line 6, 53: 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 - around 2 million rows
  - Compare their TiKV regions count and physical execution plans on PK predicts

```
// Setup: Terminal 1 - Skip this step if you are testing on TiDB Cloud
$ tiup playground v6.1.0 --tag cnc-example --db 2 --pd 3 --kv 3 --tiflash 1

// Demo Run: Terminal 2
$ git clone https://github.com/pingcap/tidb-course-201-lab.git
$ cd tidb-course-201-lab/scripts
$ ./connect-4000.sh
tidb> source 07-demo-compare-clustered-and-nonclustered-pk.sql

// Tear Down: Terminal 1 - Skip this step if you are testing on TiDB Cloud
$ <ctrl-c>
$ tiup clean cnc-example
```

K6: Clustered and Non-Clustered Primary Key (Output)

```
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`**
- Sample code:
  - Line 65, 82: 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, run it to observe the difference results:
    - **`no-retry`**: Transaction auto rolled back in front of **`ErrorCode 9007`**
    - **`retry`**: Wait and retry the failed DML if you encounter **`ErrorCode 9007`**

```
// Setup: Terminal 1 - Skip this step if you are testing on TiDB Cloud
$ tiup playground v6.1.0 --tag optimistic-example --db 2 --pd 3 --kv 3 --tiflash 1

// Demo Run: Terminal 2
$ git clone https://github.com/pingcap/tidb-course-201-lab.git
$ cd tidb-course-201-lab/scripts
$ ./09-demo-jdbc-tx-optimistic-01-show.sh no-retry
$ ./09-demo-jdbc-tx-optimistic-01-show.sh retry

// Tear Down: Terminal 1 - Skip this step if you are testing on TiDB Cloud
$ <ctrl-c>
$ tiup clean optimistic-example
```

K7: Optimistic Transaction Lock (Output) - No-retry

```
$ ./09-demo-jdbc-tx-optimistic-01-show.sh no-retry
TiDB Endpoint:127.0.0.1
TiDB Username:root
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) - Retry

```
$ ./09-demo-jdbc-tx-optimistic-01-show.sh retry
TiDB Endpoint:127.0.0.1
TiDB Username:root
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`**
- Sample code:
  - Line 61: 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

```
// Setup: Terminal 1 - Skip this step if you are testing on TiDB Cloud
$ tiup playground v6.1.0 --tag pessimistic-example --db 2 --pd 3 --kv 3 --tiflash 1

// Demo Run: Terminal 2
$ git clone https://github.com/pingcap/tidb-course-201-lab.git
$ cd tidb-course-201-lab/scripts
$ ./09-demo-jdbc-tx-pessimistic-01-show.sh
$ ./09-demo-jdbc-tx-pessimistic-01-show.sh

// Tear Down: Terminal 1 - Skip this step if you are testing on TiDB Cloud
$ <ctrl-c>
$ tiup clean pessimistic-example
```

K8: Pessimistic Transaction Lock (Output)

```
$ ./09-demo-jdbc-tx-pessimistic-01-show.sh
TiDB Endpoint:127.0.0.1
TiDB Username:root
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 Via TiCDC

- Environment: `TiUP Playground`, `Kafka`, `mysql-client`
- Preparation:
  - Download Kafka distribution
- Mini demo story:
  - `TiUP Playground` only demo
  - Start the local `Kafka` service and consumer
  - Create a `TiCDC` change capture feed 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 Via TiCDC (Demo Steps)

```
// Start Zookeeper: 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

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

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

// Start Kafka Console Consumer: 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

// Start Playground: Terminal 4
$ tiup playground v6.1.0 --tag cdc-example --db 2 --pd 3 --kv 3 --ticdc 1 --tiflash 1

// Create a TiCDC Change Feed Task: Terminal 5
$ git clone https://github.com/pingcap/tidb-course-201-lab.git
$ cd tidb-course-201-lab/scripts
$ ./13-demo-cdc-create-changefeed-01.sh

// Do Any Changes by Executing DDL/DML in Terminal 5, and observe the captured changes on Terminal 4
$ mysql -h 127.0.0.1 -P 4000 -u root
mysql> create table test.t10 (id bigint primary key);
mysql> insert into test.t10 values (100);
mysql> ...

// Tear Down: Terminal 4, 3, 2, 1
$ Press <ctrl-c> in Terminal 4, 3, 2, 1 in order
$ tiup clean cdc-example
```

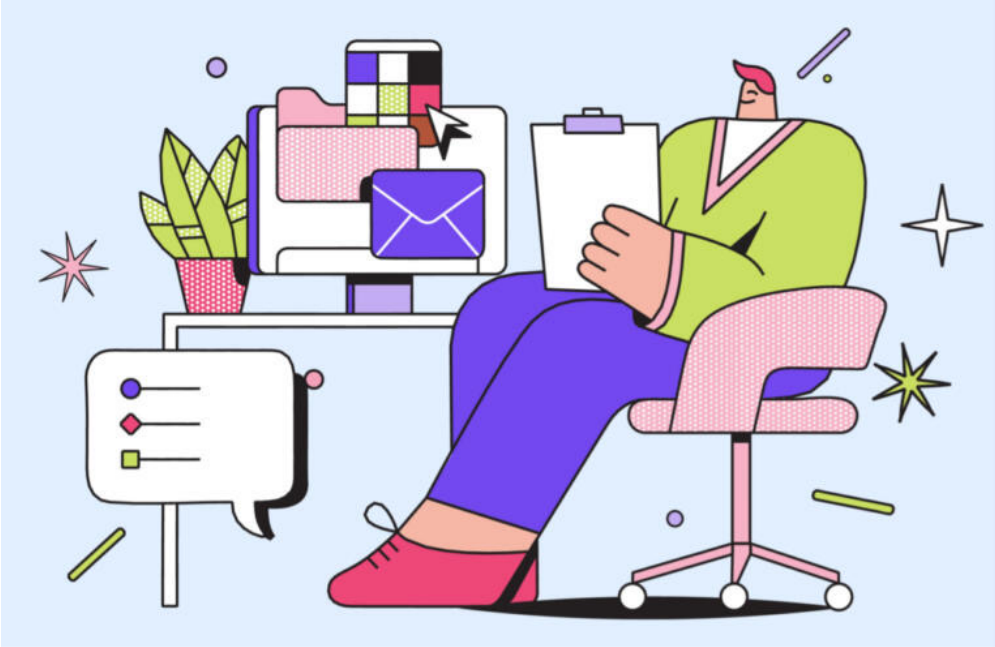
K9: Feeds Database Changes to Kafka Via TiCDC (Output)

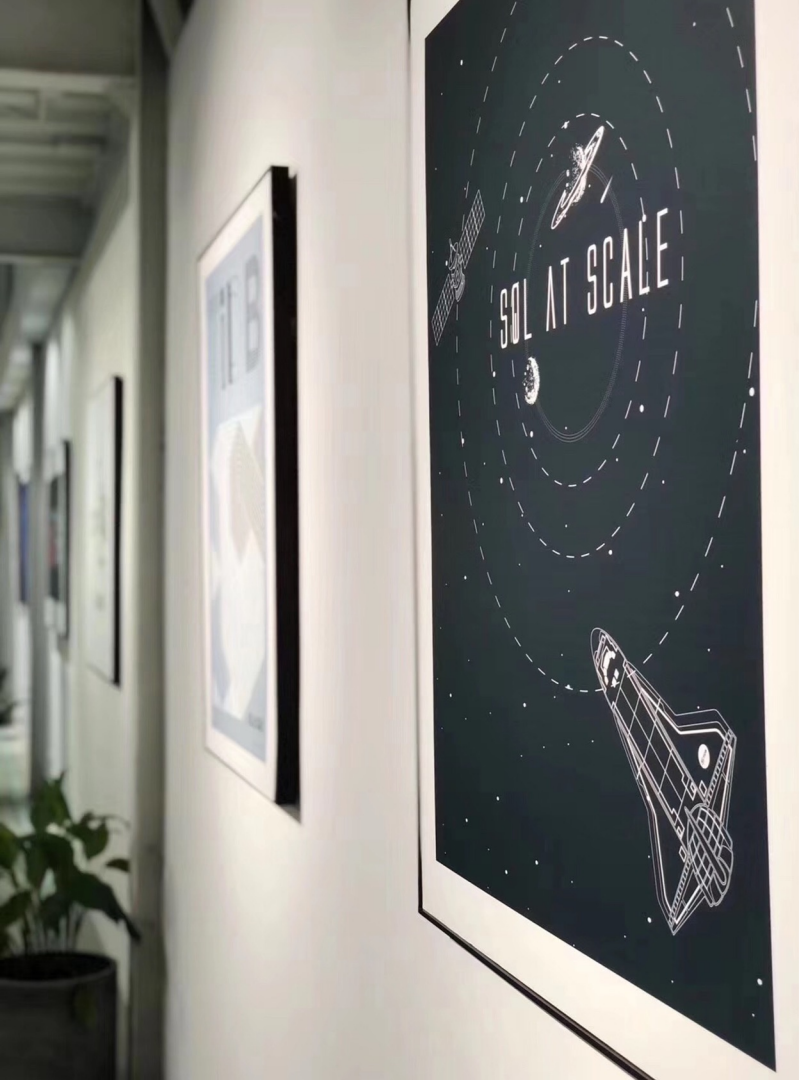
- Terminal 5 is connected to **`TiDB`**
- Terminal 3 is attached to **`Kafka Topic Consumer`**

```
// Terminal 5
mysql> create table test.t10 (id bigint primary key);
Query OK, 0 rows affected (0.25 sec)

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

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

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

// 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}}}
```

PingCAP Training & Certification: Learn TiDB from PingCAP

- Global Site: https://en.pingcap.com/education/
- China Site: https://pingcap.com/zh/education/





Thanks