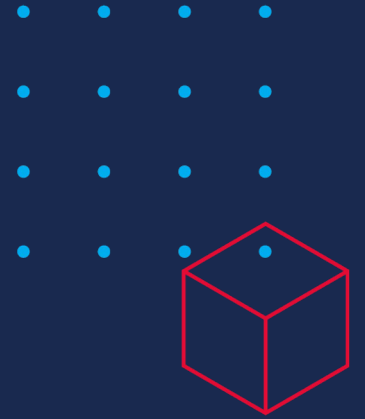

TiDB 开发者工具箱

来自 PingCAP 培训和认证: 迷你演示集合 (2022091401)

免责声明

- 本材料中的所有示例代码仅用于演示和学习目的
- 未经评估, 请勿在生产中使用它们

环境准备: TiDB Cloud 或 TiUP Playground

- 克隆演示仓库
  - `$ git clone https://github.com/pingcap/tidb-course-201-lab.git`
- **[A].** 选择 TiDB Cloud
  - 按照 TiDB Cloud Kickstart 动手训练营中的练习 1 创建 Developer Tier
  - 为 TiDB Cloud 设置环境变量:
    - `$ export TIDB_CLOUD_HOST=<hostname>`
    - `$ export TIDB_CLOUD_USERNAME=<username>`
    - `$ export TIDB_CLOUD_PASSWORD=<password>`
    - `$ export TIDB_CLOUD_PORT=<port>`
- **[B].** 或者在本地计算机上运行 **TiUP Playground** （适用于 **Linux** 或 **macOS** ）
  - 下载并安装 **TiUP** 实用程序
    - `$ curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/install.sh | sh`
    - 将命令 **tiup** 添加到 **PATH** 路径: `$ source ~/.bash_profile`
  - 启动本地沙盒 **Playground** TiDB集群
    - `$ git clone https://github.com/pingcap/tidb-course-201-lab.git`
    - `$ cd tidb-course-201-lab/scripts && ./playground-start.sh`
    - 保持终端处于打开状态,然后在另一个终端中检查 **Playground** 状态
      - `$ cd tidb-course-201-lab/scripts && ./playground-check.sh`

迷你演示清单

- **B** 部分: 最佳实践
- **K** 部分: 知识

B1: JDBC 批量插入	K1:常见数据类型的最大长度	K2:字符集（ UTF8MB4 和 GBK ）	K3: AUTO_INCREMENT
K4: AUTO_RANDOM	K5: Java 客户端 TLS 连接	K6: Clustered 和 Non-Clustered PK	K7:乐观锁
K8:悲观锁	K9:通过 TiCDC 将数据库更改为 Kafka	K10:Raw KV 示例（测试版）	B2: Python 批量插入
K11:在线 Schema 更改			

B1: JDBC 批量插入

- 环境: **`Java SDK`**
- 示例代码:
  - DemoJdbcBatchInsert.java
- 迷你演示概述:
  - 运行脚本将 **`10000`** 行插入到一个表中, 将 **`rewriteBatchedStatements`** 设置为 **`true`**。
  - 然后, 脚本将在 **`rewriteBatchedStatements`** 设置为 **`false`** 的情况下再次执行此操作。
  - 观察所用时间的差异。
- 使用参数 **`[cloud|local]`** 分别在 TiDB Cloud 或本地 Playground 上运行演示。

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

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

B1: JDBC 批量插入（输出）

- 以下输出示例来自 TiUP Playground。
- 如果客户端和 TiDB Cloud 不在同一 **Region** 中, 则两次执行之间的 **elapsed time** 差距将相当大。
  - 如果你等不及 **rewriteBatchedStatements=false** 运行完成, 可以使用 **ctrl-c**。

```
$ ./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 批量插入

- 环境: **`Python 3.9`**
- 示例代码:
  - demo-batch-insert.py
- 迷你演示概述:
  - 运行脚本, 使用批处理, 样式为 **`INSERT INTO ... VALUES (),(),(),...`**, 将 **`10000`** 行插入到表中。
  - 然后, 脚本将使用循环再次执行此操作, 并一次插入一行。
  - 观察所用时间的差异。
- 使用参数 **`[cloud|local]`** 分别在 TiDB Cloud 或本地 Playground 上运行演示。

```
// 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 批量插入（输出）

- 以下输出示例来自 **TiUP Playground**。
- 如果客户端和 TiDB Cloud 不在同一个 region, 则两次执行之间的 **elapsed time** 差距会很大。
  - 如果你等不及第二个非批量样式演示运行完成, 请使用 **ctrl-c**。

```
$ ./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: 常用数据类型的最大长度

- 环境: **Python 3.9**
- 示例代码:
  - demo-data-type-maxlength.py
- 迷你演示概述:
  - 假设字符集是 **utf8mb4**。
  - 将具有最大长度数据的行插入示例表中, 然后显示结果。
  - **TIMESTAMP** 的数据类型查询值取决于时区。
  - 以下数据类型的最大值受以下设置组合的限制:
    - TiDB: **txn-entry-size-limit** 和 **txn-total-size-limit**
    - TiKV: **raft-entry-max-size**
    - **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: 常用数据类型的最大长度（输出）

```
$ ./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: 字符集（ UTF8MB4 和 GBK ）

- 环境: **mysql-client**
- 示例代码:
  - 03-demo-charset-01-show.sql
- 迷你演示概述:
  - 使用 **CAST** 函数测试 **UTF8MB4** 和 **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> source 03-demo-charset-01-show.sql
```

K2: 字符集（ UTF8MB4 和 GBK ）（输出）

```
$ ./connect-4000.sh
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

- 环境: **mysql-client**、 **TiUP Playground**
- 示例代码:
  - 07-demo-auto-increment-01-setup.sql
  - 07-demo-auto-increment-03-show.sh
- 迷你演示概述:
  - 此演示仅限 **TiUP Playground**。
  - 使用 **AUTO_INCREMENT** 和 **AUTO_ID_CACHE 300** 创建表。
  - 从 **2** 个 TiDB server 实例中插入新行, 观察结果。

```
// 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 （输出）

```
$ ./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

- 环境: **mysql-client**
- 示例代码:
  - 07-demo-auto-random-01-show.sql
- 迷你演示概述:
  - 创建一个具有 **AUTO_RANDOM(4)** 属性的表, 插入一些行, 然后检查结果。
  - 最后一个查询应该返回 **n** 行, 而 **n** 接近 **2^4** 也就是 **16**, 为什么?

```
// 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-auto-random-01-show.sql
```

K4: AUTO_RANDOM （输出）

```
./connect-4000.sh
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: 无需服务器和客户端验证的 Java TLS 连接

- 环境: **Java SDK**
- 示例代码:
  - auto-tls 的 TiDB 示例：tls.toml
  - DemoJdbcConnectionSecured.java
- 迷你演示概述:
  - 在启用 **auto-tls** 的情况下创建 Playground （在 TiDB Cloud 案例中跳过此步骤）。
  - 设置多个 **sslMode**, 连接到 TiDB 服务器实例, 观察其中的区别。

```
// 1. Stop the default Playground you started previously in Terminal 1 - Skip this step if you are testing on TiDB Cloud
$ <ctrl-c>

// 2. Start a TLS enabled Playground in Terminal 1 - Skip this step if you are testing on TiDB Cloud
$ ./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 cloud|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 - Skip this step if you are testing on TiDB Cloud
$ ./playground-clean-classroom-tls.sh
$ ./playground-start.sh
```

K5: 无需服务器和客户端验证的 Java TLS 连接（输出）

```
$ ./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 ###
...
        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 和 Non-Clustered 主键

- 环境: **`mysql-client`**
- 示例代码:
  - 07-demo-compare-clustered-and-nonclustered-pk.sql
- 迷你演示概述:
  - 使用 **`Clustered`** PK 创建表 1。
  - 使用 **`Non-Clustered`** PK 创建表 2, 从表 1 复制数据。
  - 两个表的数据量相似 —— 大约 200 万行。
  - 比较它们的 Region 数量，以及在相同 WHERE 语句下不同的物理执行计划

```
// 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 和 Non-Clustered 主键（输出）

```
$ ./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: 乐观锁

- 环境: **Java SDK**
- 示例代码:
  - DemoJdbcTxOptimisticLock.java
- 迷你演示概述:
  - 在 **optimistic mode** 中, 两个事务同时修改同一行可能会导致冲突。
  - 该脚本提供了两个选项, 运行它来观察差异结果:
    - **no-retry**: 事务在 **ErrorCode 9007** 发生时自动回滚。
    - **retry**: 如果事务遇到 **ErrorCode 9007**, 等待并重试失败的 DML。

```
// 1. Go to 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: 乐观锁（输出）- no-retry

```
$ ./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▧▧▧
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: 乐观锁（输出）- no-retry

```
$ ./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▧▧▧
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: 悲观锁

- 环境: **Java SDK**
- 示例代码:
  - DemoJdbcPessimisticLock.java
- 迷你演示概述:
  - 在 **pessimistic mode** 中, 两个事务同时修改同一行不会导致冲突。
  - 被阻塞的会话将等待事务锁释放。
  - 没有 **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: 悲观锁（输出）

```
$ ./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▧▧▧
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: 通过 TiCDC 将数据库更改同步到 Kafka

- 环境: **TiUP Playground**、 **Kafka**、 **mysql-client**
- 准备:
  - 下载 Kafka 发行版
- 迷你演示概述:
  - 此演示仅限 **TiUP Playground**。
  - 启动本地 **Kafka** service and consumer。
  - 使用 **open-protocol** 创建 **TiCDC** 变更捕捉同步任务（其他可用协议）。
  - 随意执行 **DDL**、 **DML**, 并观察 **Kafka** consumer 端捕捉的变更。

K9: 通过 TiCDC 将数据库更改同步到 Kafka （演示步骤）

```
// 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 v6.1.0 --tag cdc-example --db 2 --pd 3 --kv 3 --ticdc 1 --tiflash 1

// 7. 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

// 8. 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> ...

// 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: 通过 TiCDC 将数据库更改同步到 Kafka （输出示例）

- 终端 5 连接到 **TiDB**
- 终端 3 连接到 **Kafka Topic Consumer**

```
// On terminal 5, execute CREATE/INSERT/UPDATE/DELETE in order
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)

// 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: 原始 KV 示例

- 环境: **Python 3.x**
- 示例代码:
  - demo-simple-put-get-rawkv.py
- 迷你演示概述:
  - 此演示仅限 **TiUP Playground**。

```
// 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: 原始 KV 示例（输出）

```
$ ./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: 在线 Schema 更改

- 环境: `Java SDK`、 `mysql-client`
- 示例代码:
  - DemoJdbcPreparedStatement8028.java
  - 07-demo-online-ddl-add-column-02.sql
- 迷你演示概述:
  - 会话 A 运行 workload 以插入行, 总计 192000 行。
    - Workload 脚本: `11-demo-jdbc-prepared-statement-online-ddl-01-show.sh`
  - 另一个会话执行 DDL, 为会话 A 中正在插入行的表添加新列。
    - 注意: DML 不会阻止 TiDB 中的 DDL, 反之亦然。
  - 在第一次演示运行中, 在没有错误代码提示的情况下执行 workload, 观察在线 DDL 如何影响会话 A 的 DML 的结果。
  - 在第二次演示运行中, 以 `8028` 作为第二个参数执行 workload, 设置程序在遇到错误代码 8028 时重新执行事务。
    - 错误代码 8028: `Information schema is changed during the execution of the statement`
- 使用参数 `[cloud|local]` 分别在 TiDB Cloud 或本地 Playground 上运行演示。
- 下一张幻灯片中列出了详细的演示运行步骤...

K11: 在线 Schema 更改

```
// 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> source 07-demo-online-ddl-add-column-02.sql

      // Observe what happened in terminal 1, 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> source 07-demo-online-ddl-add-column-02.sql

      // Observe what happened in terminal 1, how many rows had been inserted?
```

K11: 首次演示运行（输出）

```
// 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
...
/* Executing query: SELECT name1 as "|NAME1|", count(*) as "|BEFORE-DDL-GOAL: 192000|" FROM test.target_table GROUP BY name1 ORDER BY 1; */
        Row#, name1, |BEFORE-DDL-GOAL: 192000|
        1) BEFORE-DDL, 960
...
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: 第二次演示运行（输出）

```
// 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
...
/* Executing query: SELECT name1 as "|NAME1|", count(*) as "|BEFORE-DDL-GOAL: 192000|" FROM test.target_table GROUP BY name1 ORDER BY 1; */
        Row#, name1, |BEFORE-DDL-GOAL: 192000|
        1) BEFORE-DDL, 960
...
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.
...
/* Executing query: SELECT name1 as "|NAME1|", count(*) as "|BEFORE-DDL-GOAL: 192000|" FROM test.target_table GROUP BY name1 ORDER BY 1; */
        Row#, name1, |BEFORE-DDL-GOAL: 192000|
        1) BEFORE-DDL, 192000
I: 199
Turn on autocommit.
Connection closed.
```

PingCAP Training and Certification: Learn TiDB from PingCAP

- TiDB 文档
- TiDB Cloud 文档
- PingCAP 培训和认证门户

Thanks