ORACLE

FOSDEM 2022 MySQL Devroom

MySQL 8.0: Logical Backups, Snapshots and PITR like a rockstar

Frédéric Descamps
Community Manager
MySQL
February 2022

Who am I ?

about.me/lefred

Frédéric Descamps

- *@lefred*
- *MySQL Evangelist*
- *hacking MySQL since 3.21*
- *devops believer*
- *living in* 🇧🇪
- *https://lefred.be*

2022 best practices

settings

I assume that your system

- *is running MySQL* ***8.0.27*** *or later*
- *uses only InnoDB*
- *has binary logs enabled (required for PITR)*
- *binary logs must use* ***ROW*** *format*
- *uses GTID*

Point-in-Time Recovery

using the binary logs

Point-in-Time Recovery

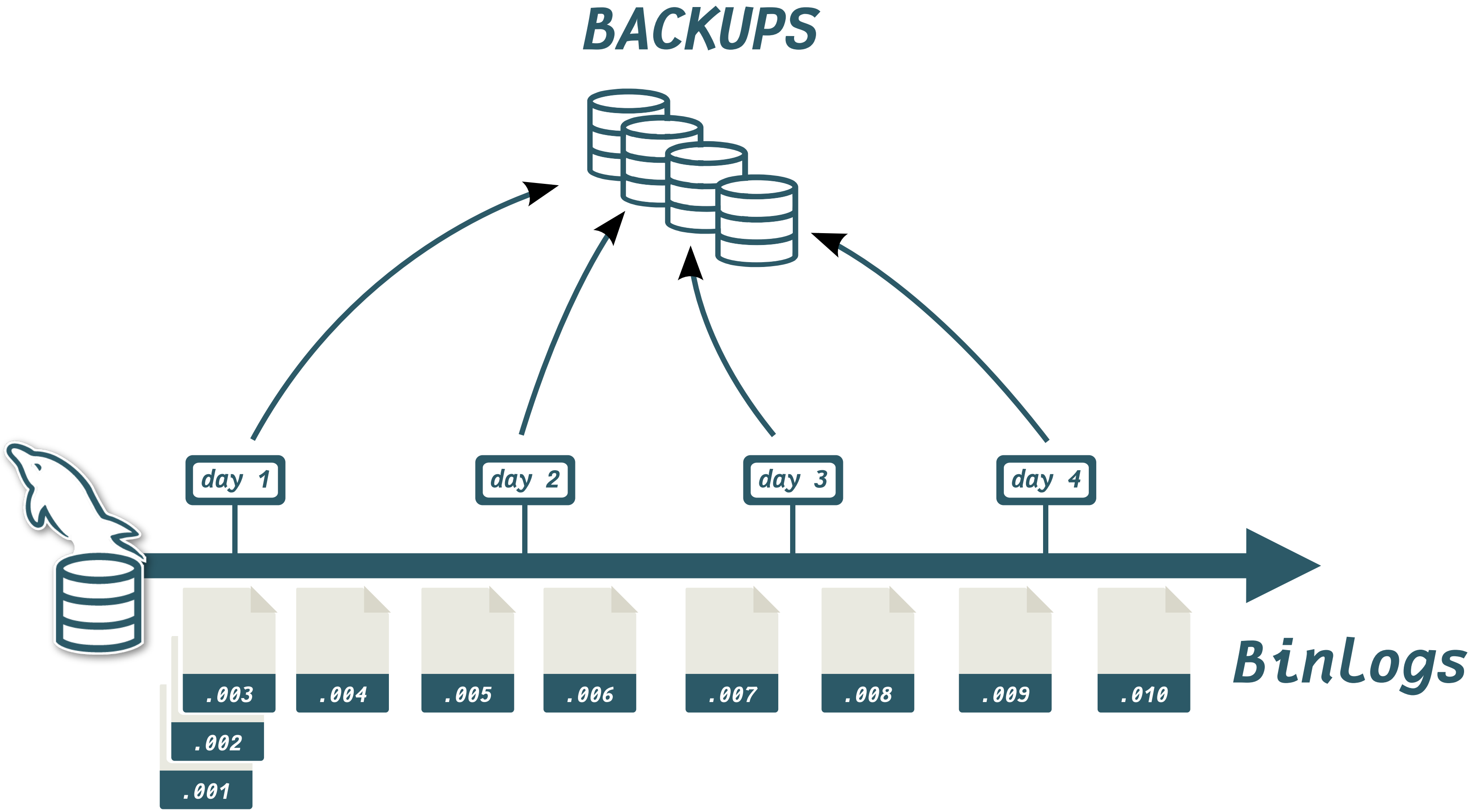
This is the technique whereby an administrator can restore or recover a set of data to a certain point usually in the past.

In MySQL, point-in-time recovery consists in restoring a dump of the data and then replay the binary logs from and to a specific point.

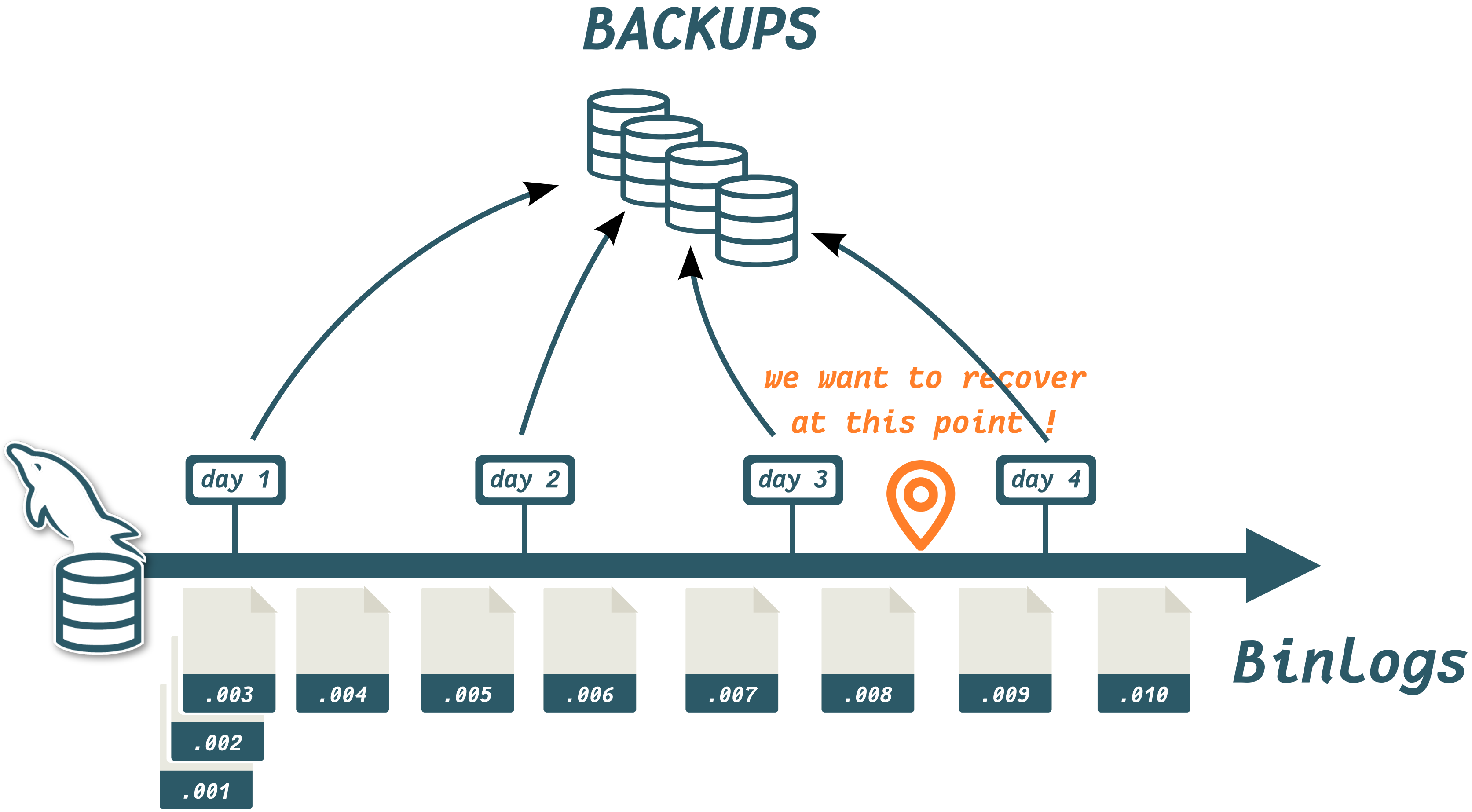
This technique is used for:

- *fixing a problem*
- *live migration*

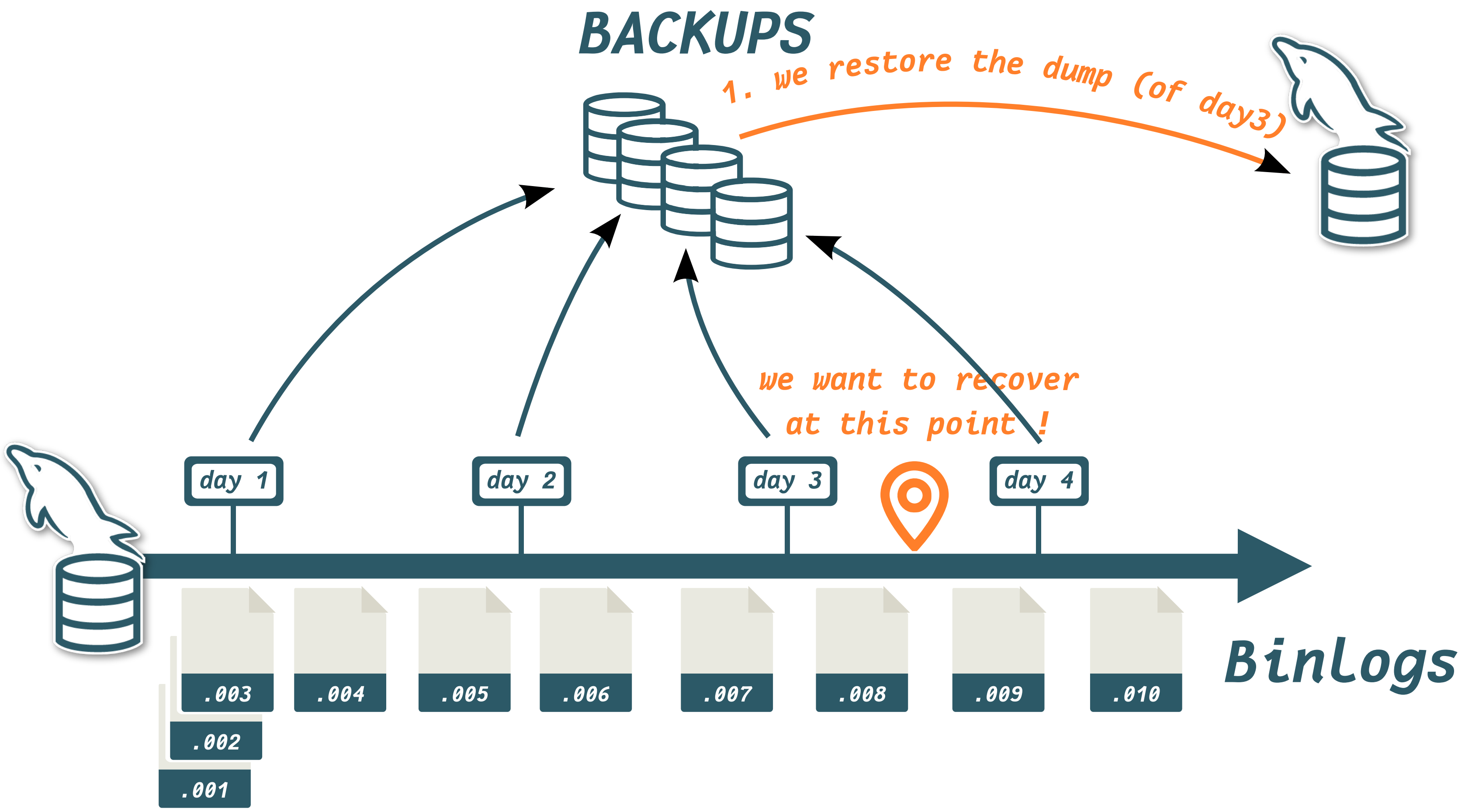
Point-in-Time Recovery : how does it work ?



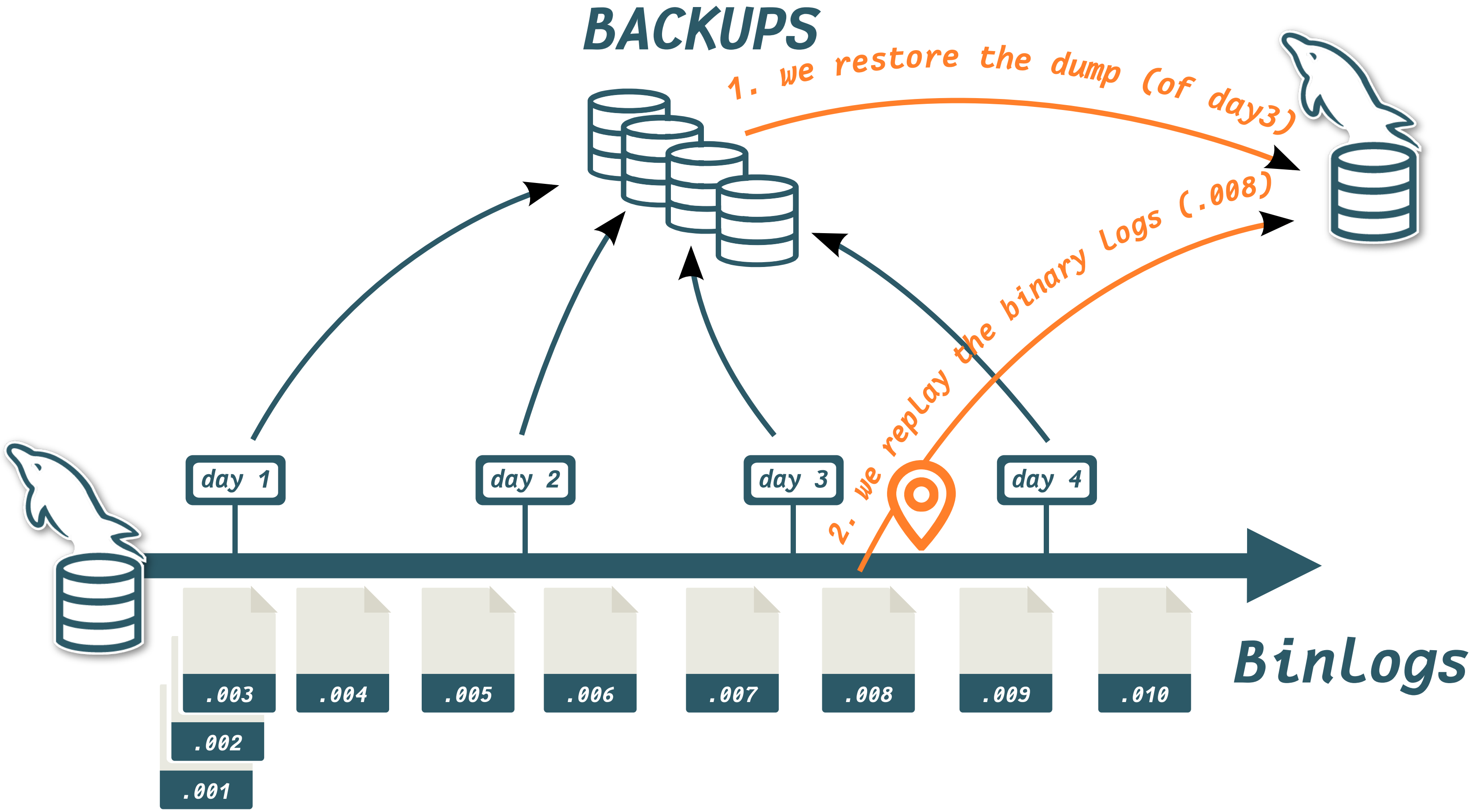
Point-in-Time Recovery : how does it work ?



Point-in-Time Recovery : how does it work ?

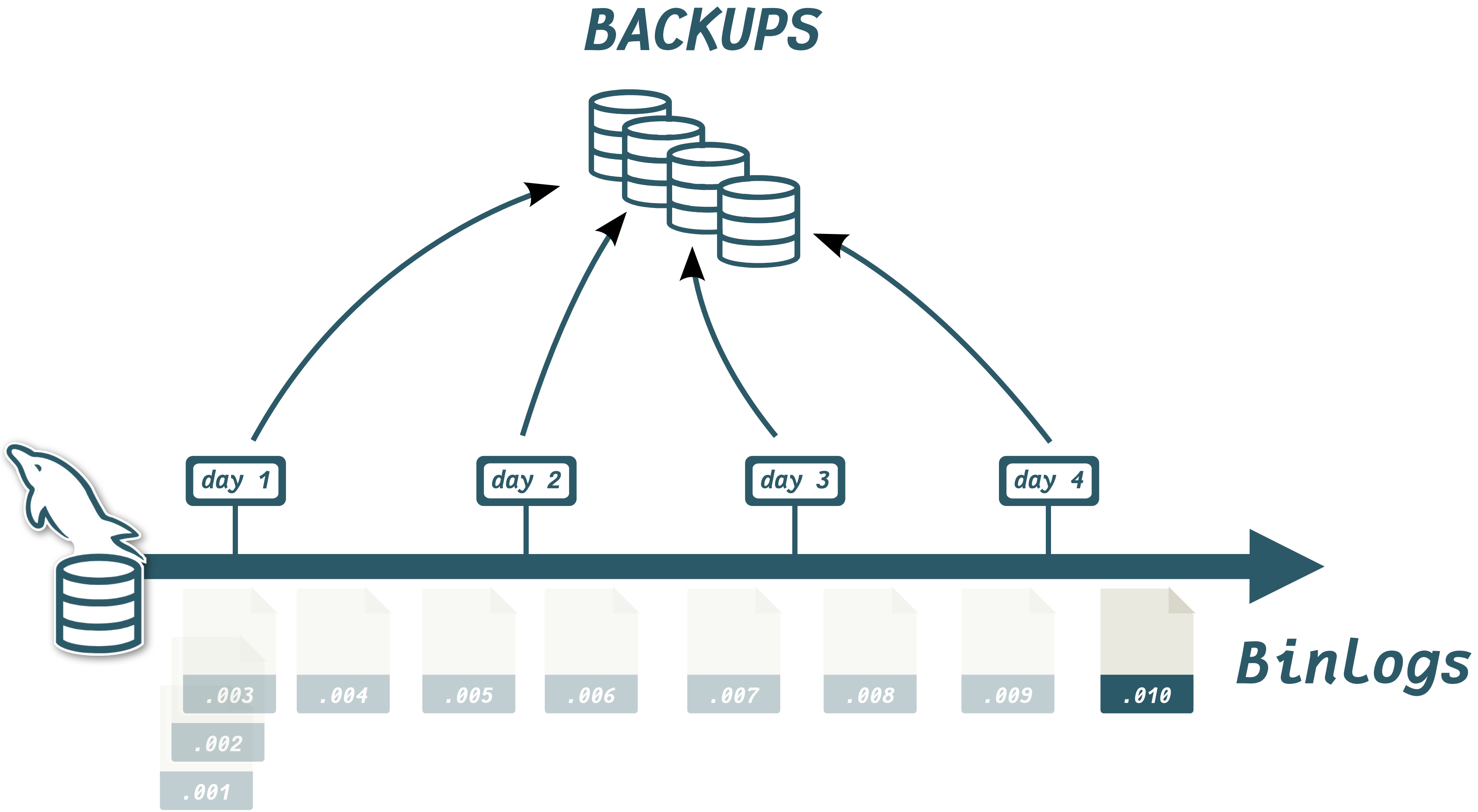


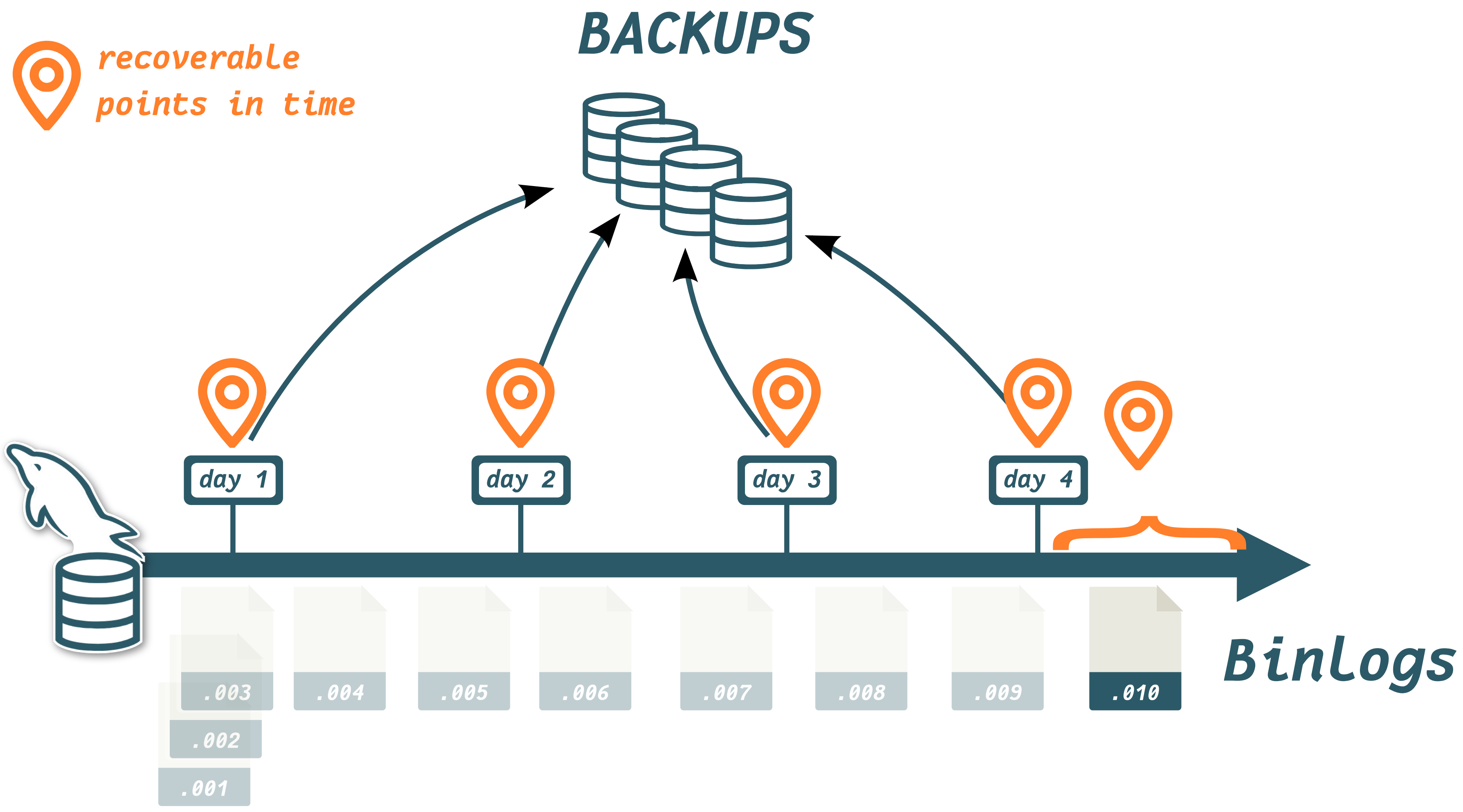
Point-in-Time Recovery : how does it work ?



Point-in-Time Recovery : important concept

Usually after a backup is made and verified, binary log files are purged from the MySQL server:


BACKUPS
day 1
day 2
day 3
day 4
.003
.002
.001
.004
.005
.006
.007
.008
.009
.010
Binlogs

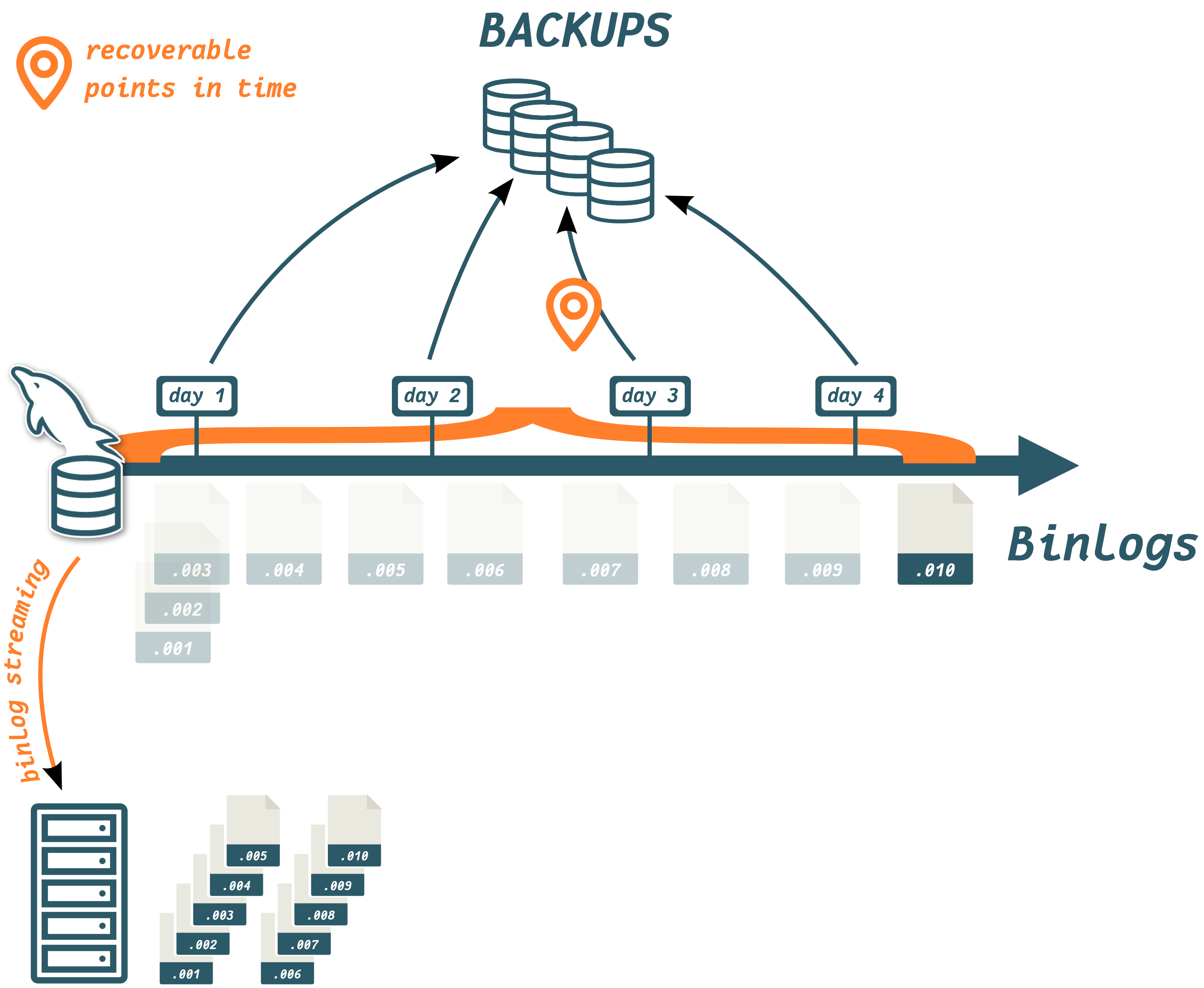

recoverable
points in time
BACKUPS
day 1
day 2
day 3
day 4
.003
.004
.005
.006
.007
.008
.009
.010
.002
.001
Binlogs

Point-in-Time Recovery : important concept (2)

As you can see we can only recover to the exact time of the backups/dumps and do point-in-time recovery only from the last one !

This is why it's recommended to also stream the binary logs somewhere else (another server, a NAS, the cloud, ...).

This will allow to make a point-in-time recovery at any point back in time:


recoverable points in time
BACKUPS
day 1
day 2
day 3
day 4
Binlogs
.001
.002
.003
.004
.005
.006
.007
.008
.009
.010
binlog streaming
.001
.002
.003
.004
.005
.006
.007
.008
.009
.010

Point-in-Time Recovery for Fixing Something

Why should we perform point-in-time recovery ?

Point-in-Time Recovery for Fixing Something

Why should we perform point-in-time recovery ?

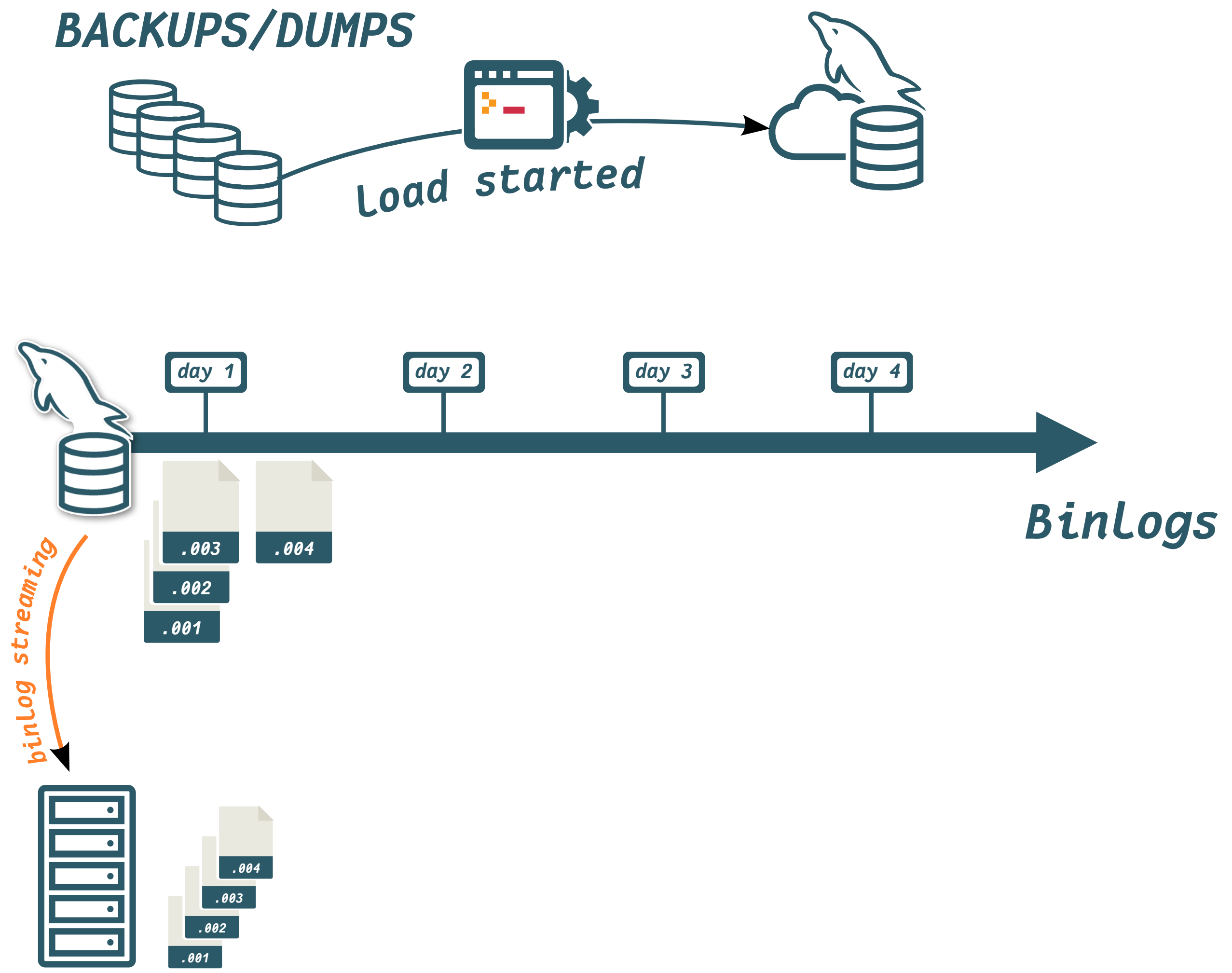
- *a user made a mistake*
- *we need to find back data from a certain point-in-time*
- *we need to have an overview of the database at a certain time*

Point-in-Time Recovery for Live Migration

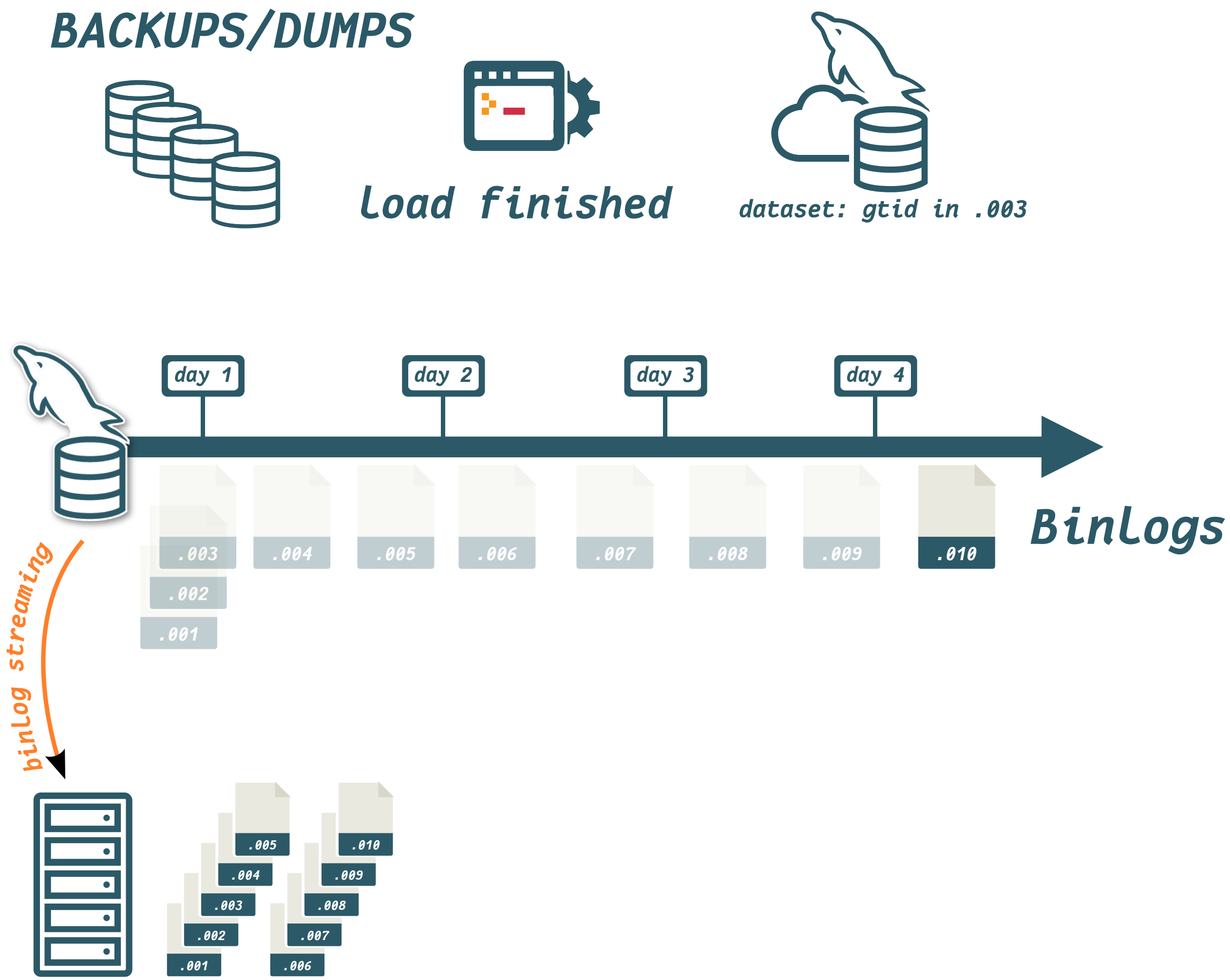
When we do large migration (to the cloud for example), the load time can take longer than the binary log retention on the MySQL server that will be used as Replication Source.

Then, Point-in-time recovery technique will be used to sync the future replica to be elligible for asynchronous replication.

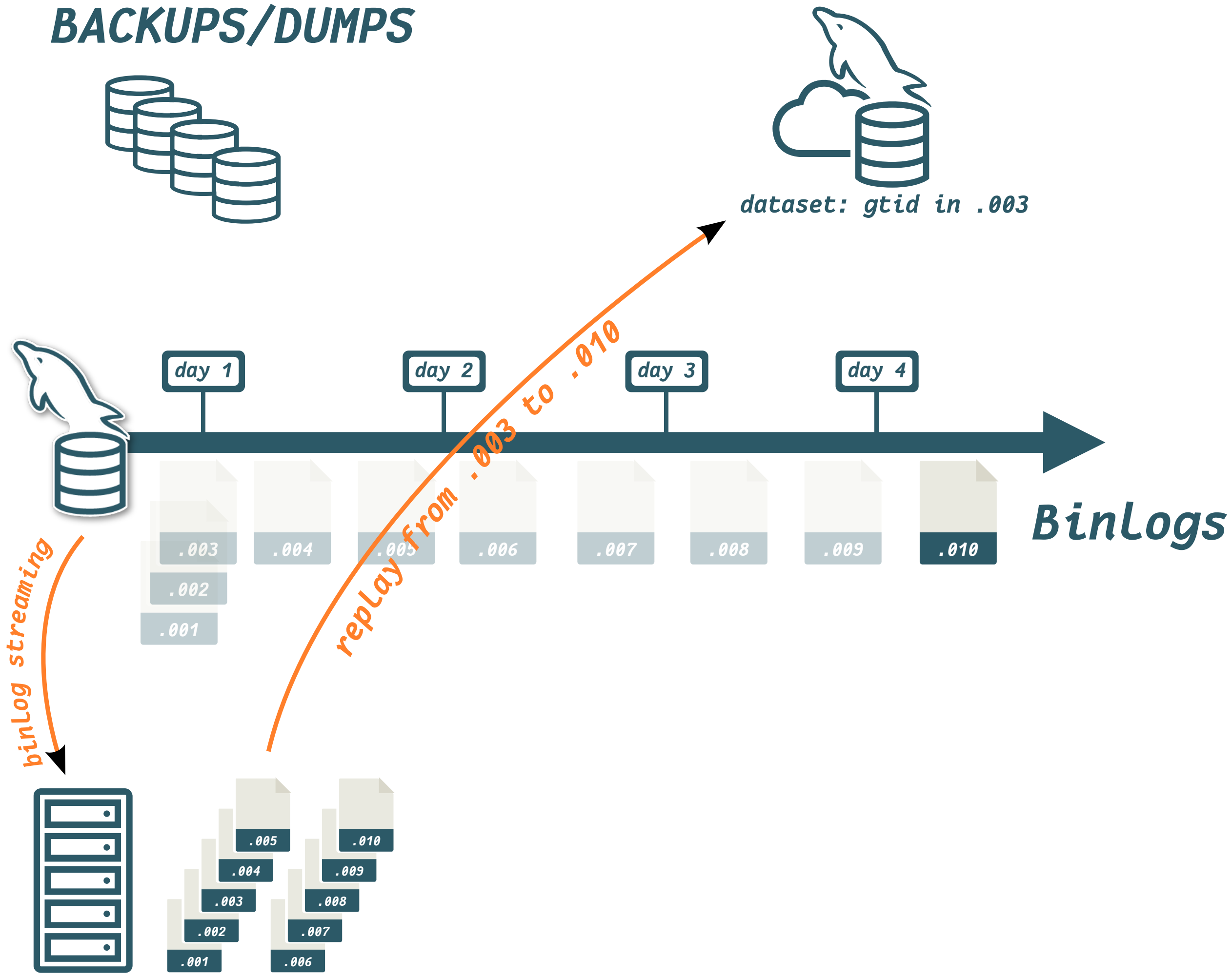
Point-in-Time Recovery for Live Migration



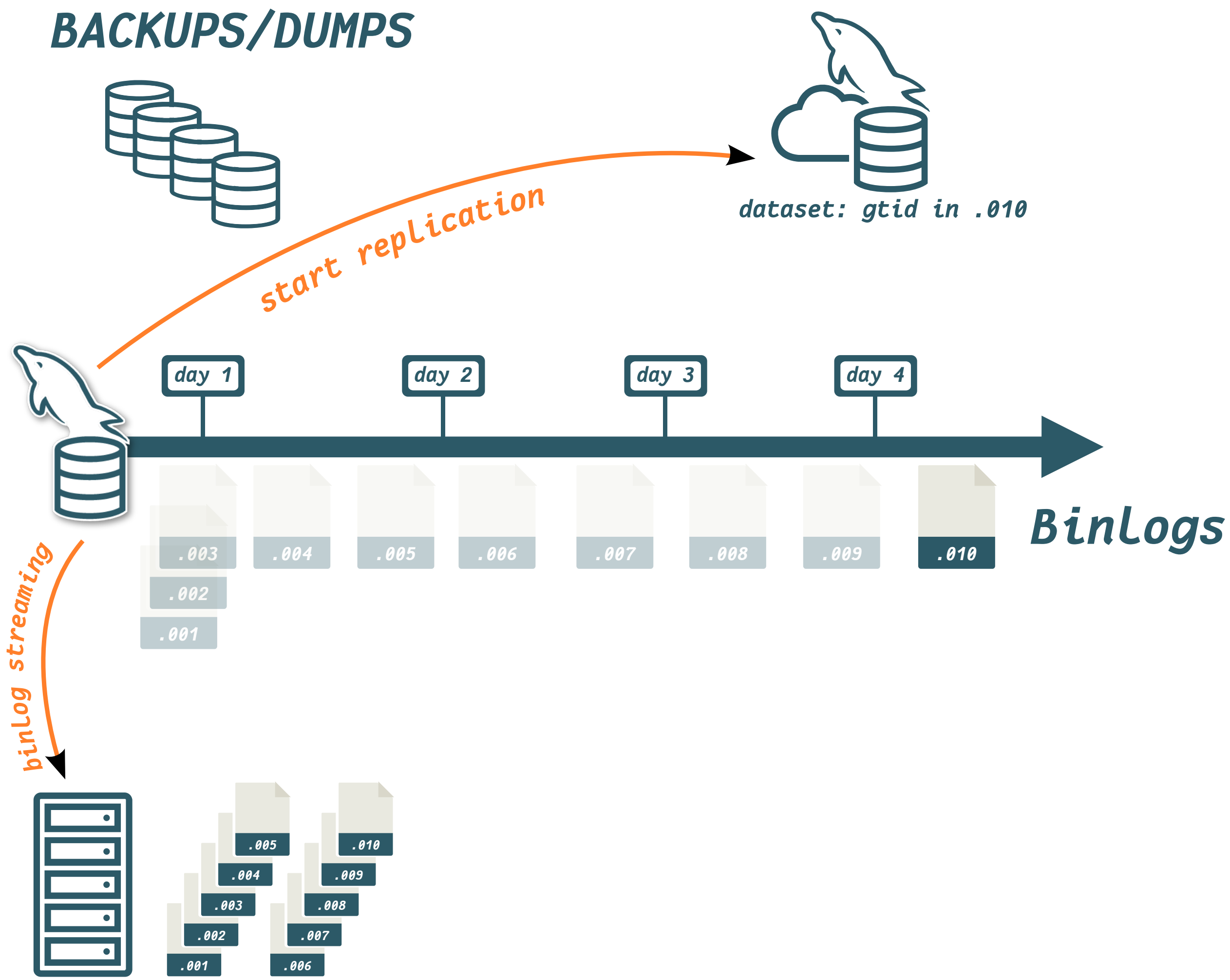
Point-in-Time Recovery for Live Migration



Point-in-Time Recovery for Live Migration



Point-in-Time Recovery for Live Migration



Backups

Physical, Logical, Snapshot, …

Backups

For years, physical hot backups were recommended. With the increase in use of the cloud for MySQL, logical backups are coming back to the forefront.

Backups

For years, physical hot backups were recommended. With the increase in use of the cloud for MySQL, logical backups are coming back to the forefront.

First with **`mysqldump`***... but as you may know, this tool is not optimal. Single-threaded to dump* ***and*** *single-threaded to load the data.*

Backups

For years, physical hot backups were recommended. With the increase in use of the cloud for MySQL, logical backups are coming back to the forefront.

First with `mysqldump`*... but as you may know, this tool is not optimal. Single-threaded to dump* ***and*** *single-threaded to load the data.*

That's why, Oracle came up with MySQL Shell ***Dump & Load Utility*** *!*

MySQL Shell Dump & Load Utility

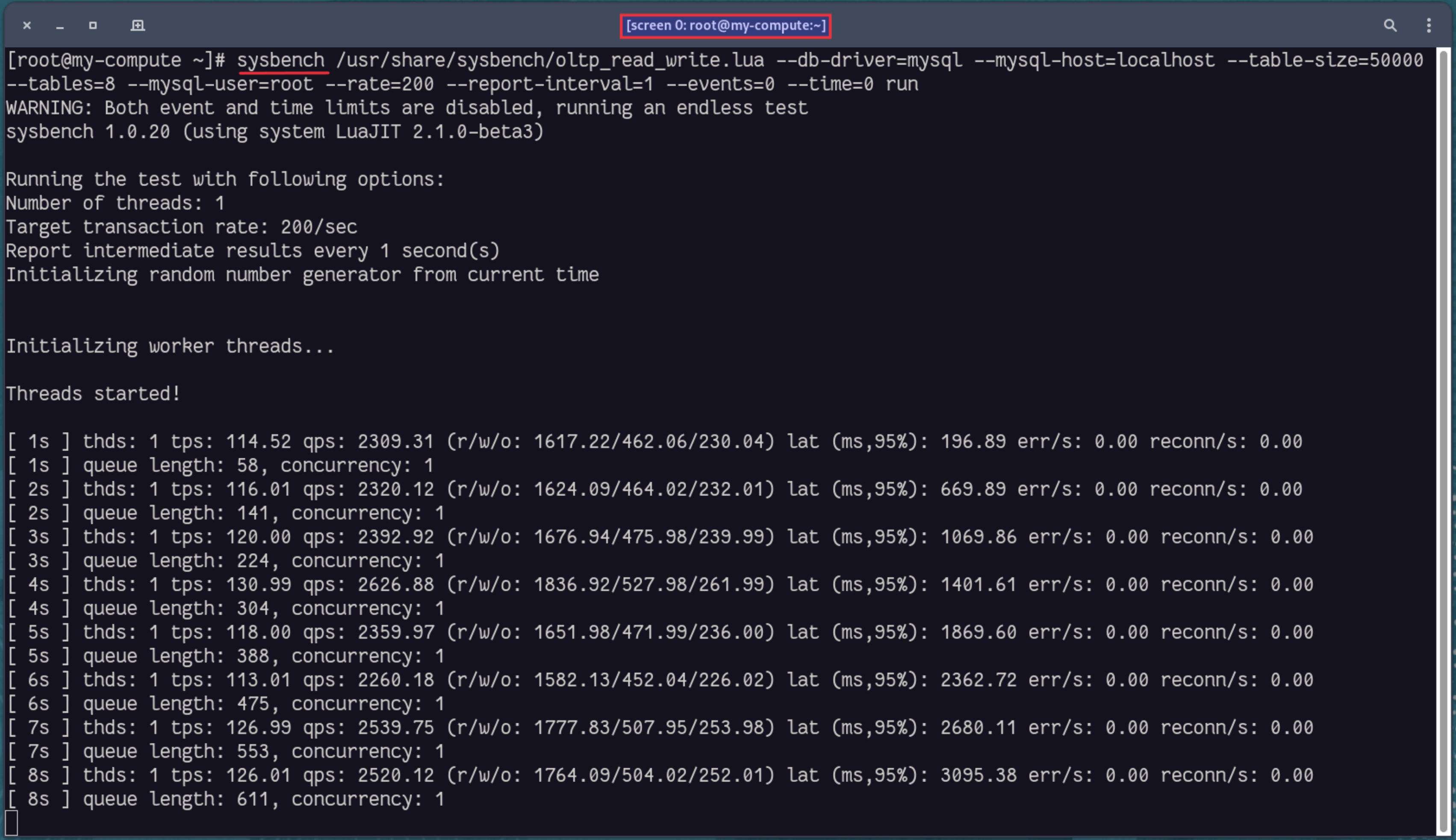
- *introduced with MySQL 8.0.21*
- *supports export of all or selected schema*
- *supports local storage (could be a mount to S3) and OCI Object Storage natively*
- *supports dump from 5.6 (since 8.0.26), 5.7 and 8.0*
- *can "fix" your schema (force InnoDB, add an invisible primary key, ...)*
- *dumps and loads in parallel*
- *and more ...*

The environment

To illustrate the scenarios in this presentation, I use the following system:

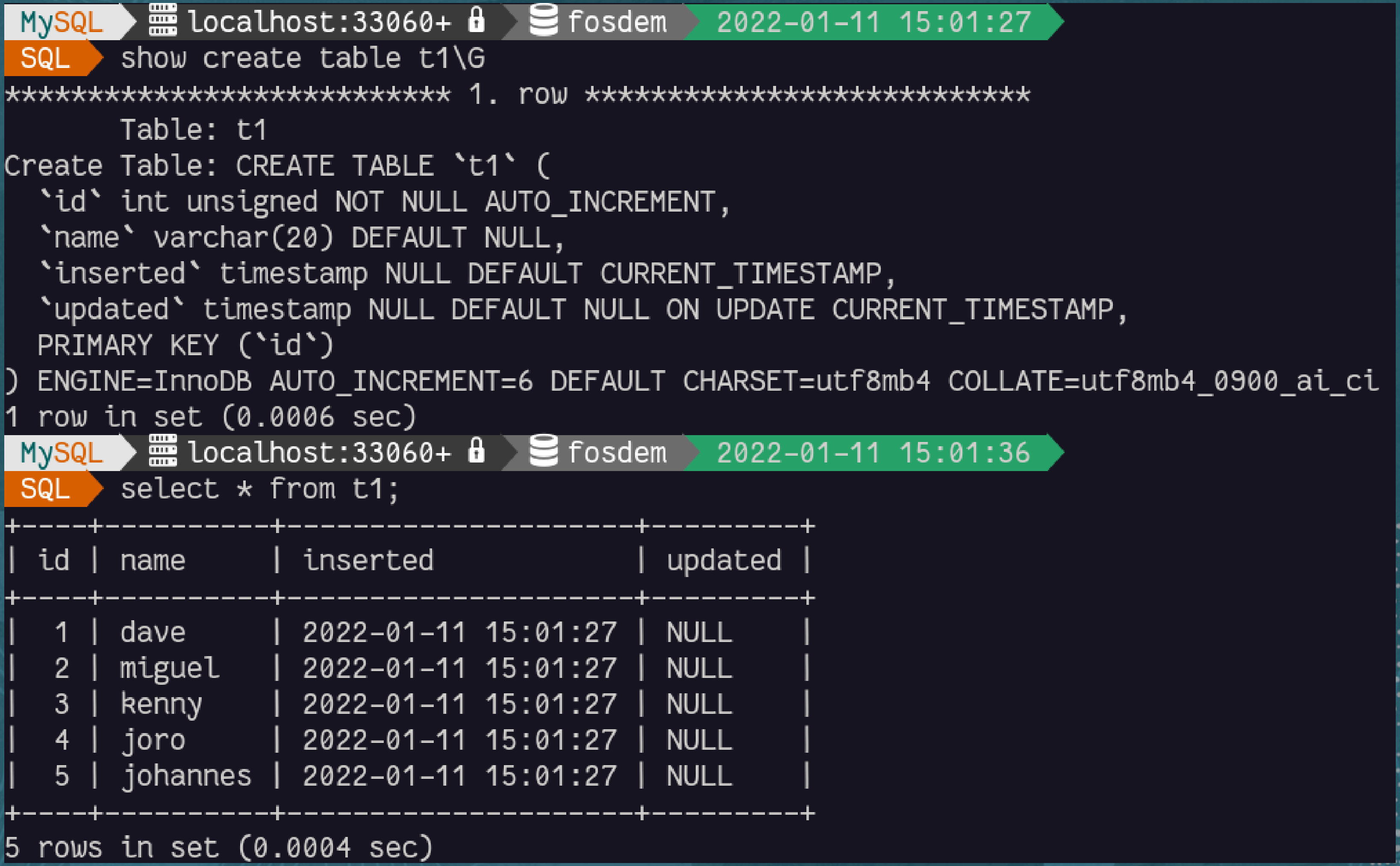
- *Ampere compute instance (VM.Standard.A1.Flex, 4 OCPU, 24GB RAM)*
- *MySQL Server 8.0.27*
- *MySQL Shell 8.0.27*
- *sysbench 1.0.20 (generating load and data)*
- *a specify table to play:* `fosdem.t1`

Sysbench



```
[root@my-compute ~]# sysbench /usr/share/sysbench/oltp_read_write.lua --db-driver=mysql --mysql-host=localhost --table-size=50000
--tables=8 --mysql-user=root --rate=200 --report-interval=1 --events=0 --time=0 run
WARNING: Both event and time limits are disabled, running an endless test
sysbench 1.0.20 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 1
Target transaction rate: 200/sec
Report intermediate results every 1 second(s)
Initializing random number generator from current time


Initializing worker threads...

Threads started!

[ 1s ] thds: 1 tps: 114.52 qps: 2309.31 (r/w/o: 1617.22/462.06/230.04) lat (ms,95%): 196.89 err/s: 0.00 reconn/s: 0.00
[ 1s ] queue length: 58, concurrency: 1
[ 2s ] thds: 1 tps: 116.01 qps: 2320.12 (r/w/o: 1624.09/464.02/232.01) lat (ms,95%): 669.89 err/s: 0.00 reconn/s: 0.00
[ 2s ] queue length: 141, concurrency: 1
[ 3s ] thds: 1 tps: 120.00 qps: 2392.92 (r/w/o: 1676.94/475.98/239.99) lat (ms,95%): 1069.86 err/s: 0.00 reconn/s: 0.00
[ 3s ] queue length: 224, concurrency: 1
[ 4s ] thds: 1 tps: 130.99 qps: 2626.88 (r/w/o: 1836.92/527.98/261.99) lat (ms,95%): 1401.61 err/s: 0.00 reconn/s: 0.00
[ 4s ] queue length: 304, concurrency: 1
[ 5s ] thds: 1 tps: 118.00 qps: 2359.97 (r/w/o: 1651.98/471.99/236.00) lat (ms,95%): 1869.60 err/s: 0.00 reconn/s: 0.00
[ 5s ] queue length: 388, concurrency: 1
[ 6s ] thds: 1 tps: 113.01 qps: 2260.18 (r/w/o: 1582.13/452.04/226.02) lat (ms,95%): 2362.72 err/s: 0.00 reconn/s: 0.00
[ 6s ] queue length: 475, concurrency: 1
[ 7s ] thds: 1 tps: 126.99 qps: 2539.75 (r/w/o: 1777.83/507.95/253.98) lat (ms,95%): 2680.11 err/s: 0.00 reconn/s: 0.00
[ 7s ] queue length: 553, concurrency: 1
[ 8s ] thds: 1 tps: 126.01 qps: 2520.12 (r/w/o: 1764.09/504.02/252.01) lat (ms,95%): 3095.38 err/s: 0.00 reconn/s: 0.00
[ 8s ] queue length: 611, concurrency: 1
```

Table t1



```
MySQL  localhost:33060+  fosdem  2022-01-11 15:01:27
SQL  show create table t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `id` int unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(20) DEFAULT NULL,
  `inserted` timestamp NULL DEFAULT CURRENT_TIMESTAMP,
  `updated` timestamp NULL DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
1 row in set (0.0006 sec)
MySQL  localhost:33060+  fosdem  2022-01-11 15:01:36
SQL  select * from t1;
+----+----------+---------------------+---------+
| id | name     | inserted            | updated |
+----+----------+---------------------+---------+
|  1 | dave     | 2022-01-11 15:01:27 | NULL    |
|  2 | miguel   | 2022-01-11 15:01:27 | NULL    |
|  3 | kenny    | 2022-01-11 15:01:27 | NULL    |
|  4 | joro     | 2022-01-11 15:01:27 | NULL    |
|  5 | johannes | 2022-01-11 15:01:27 | NULL    |
+----+----------+---------------------+---------+
5 rows in set (0.0004 sec)
```

Logical Dump

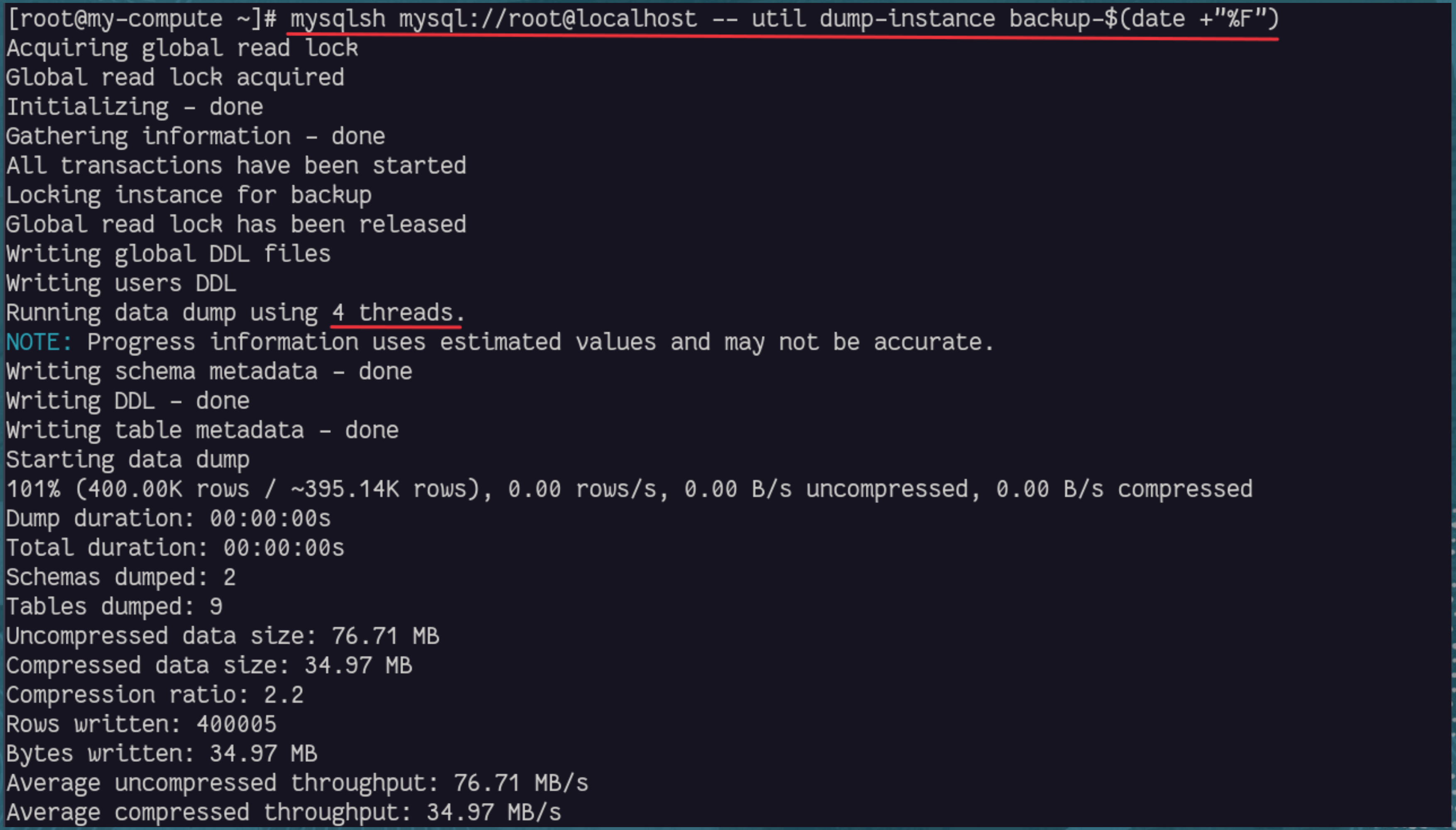
As we plan to use our logical dump as a backup (or at least as an initial dump), we won't focus on a specific schema but dump the full instance using **`util.dumpInstance()`***:*

Logical Dump

As we plan to use our logical dump as a backup (or at least as an initial dump), we won't focus on a specific schema but dump the full instance using `util.dumpInstance()`*:*

```
mysqlsh mysql://root@localhost -- util dump-instance backup-$(date +"%F")
```

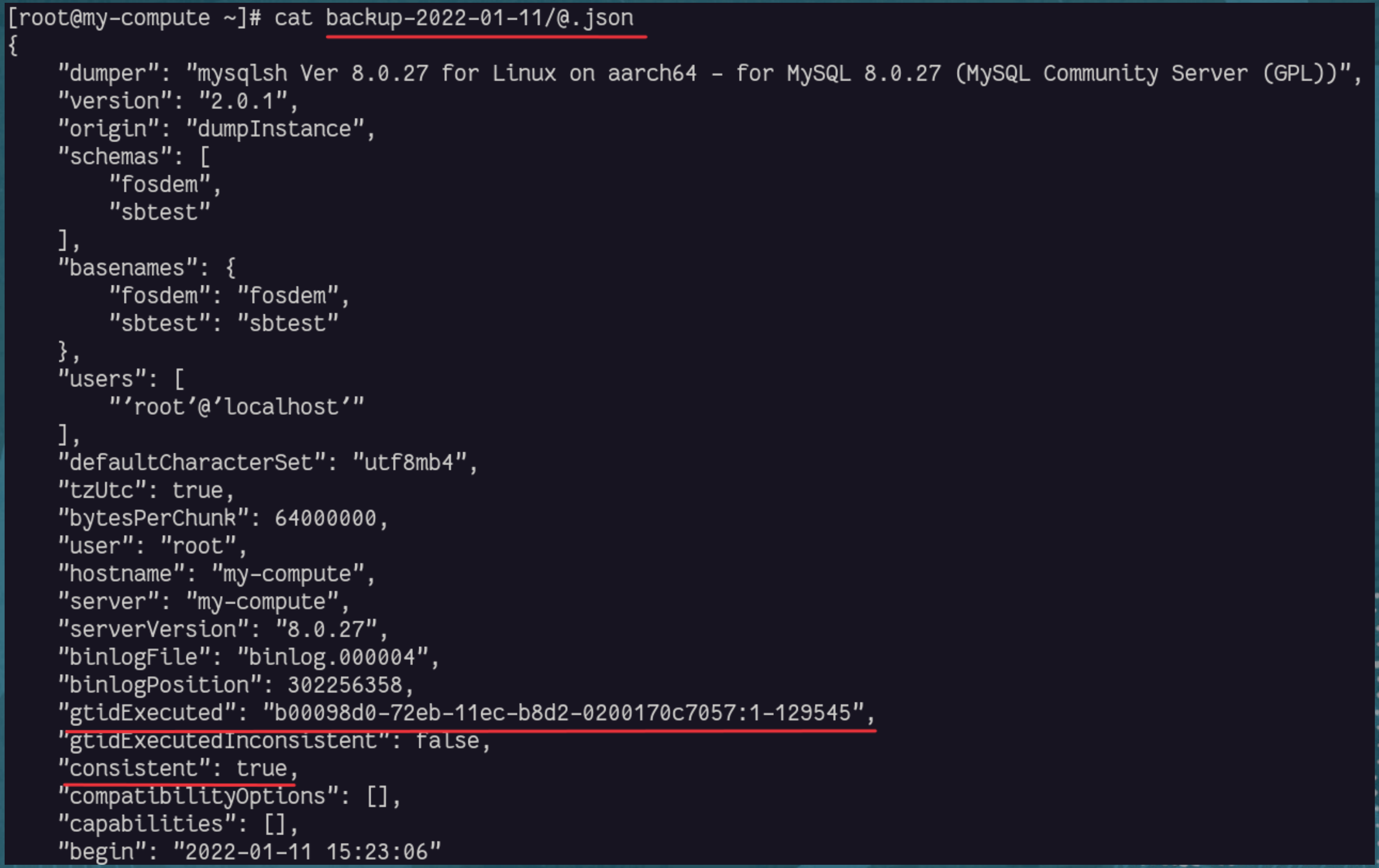
Logical Dump (2)



```
[root@my-compute ~]# mysqlsh mysql://root@localhost -- util dump-instance backup-$(date +"%F")
Acquiring global read lock
Global read lock acquired
Initializing - done
Gathering information - done
All transactions have been started
Locking instance for backup
Global read lock has been released
Writing global DDL files
Writing users DDL
Running data dump using 4 threads.
NOTE: Progress information uses estimated values and may not be accurate.
Writing schema metadata - done
Writing DDL - done
Writing table metadata - done
Starting data dump
101% (400.00K rows / ~395.14K rows), 0.00 rows/s, 0.00 B/s uncompressed, 0.00 B/s compressed
Dump duration: 00:00:00s
Total duration: 00:00:00s
Schemas dumped: 2
Tables dumped: 9
Uncompressed data size: 76.71 MB
Compressed data size: 34.97 MB
Compression ratio: 2.2
Rows written: 400005
Bytes written: 34.97 MB
Average uncompressed throughput: 76.71 MB/s
Average compressed throughput: 34.97 MB/s
```

Logical Dump (3) - metadata

The metadata of the dump is a very important file called `@.json` *and it's located in the dump's directory:*



```
[root@my-compute ~]# cat backup-2022-01-11/@.json
{
    "dumper": "mysqlsh Ver 8.0.27 for Linux on aarch64 - for MySQL 8.0.27 (MySQL Community Server (GPL))",
    "version": "2.0.1",
    "origin": "dumpInstance",
    "schemas": [
        "fosdem",
        "sbtest"
    ],
    "basenames": {
        "fosdem": "fosdem",
        "sbtest": "sbtest"
    },
    "users": [
        "’root’@’localhost’"
    ],
    "defaultCharacterSet": "utf8mb4",
    "tzUtc": true,
    "bytesPerChunk": 64000000,
    "user": "root",
    "hostname": "my-compute",
    "server": "my-compute",
    "serverVersion": "8.0.27",
    "binlogFile": "binlog.000004",
    "binlogPosition": 302256358,
    "gtidExecuted": "b00098d0-72eb-11ec-b8d2-0200170c7057:1-129545",
    "gtidExecutedInconsistent": false,
    "consistent": true,
    "compatibilityOptions": [],
    "capabilities": [],
    "begin": "2022-01-11 15:23:06"
```

GTID - MySQL Shell Dump & Load Utility

We can see that our dump is consistent and that the last GTID part of it is:

```
"gtidExecuted": "b00098d0-72eb-11ec-b8d2-0200170c7057:1-129545",
```

GTID - MySQL Shell Dump & Load Utility

We can see that our dump is consistent and that the last GTID part of it is:

```
"gtidExecuted": "b00098d0-72eb-11ec-b8d2-0200170c7057:1-129545",
```

On the MySQL Server, we can see that `sysbench` *is still running and keeps generating data:*

```
 SQL > select @@gtid_executed;
+-----------------------------------------------+
| @@gtid_executed                               |
+-----------------------------------------------+
| b00098d0-72eb-11ec-b8d2-0200170c7057:1-296244 |
+-----------------------------------------------+
```

Snapshots

physical hot dumps

Physical Hot Snapshots

There are multiple ways of doing Snapshots:

- *Hot Backups (MEB, Xtrabackup): plenty of features, can be complicated to operate*
- *Filesystem snapshots: not always hot depending on the technique and the filesystem used.*
- *MySQL CLONE*

CLONE

Clone, introduced in MySQL 8.0.17, permits cloning data locally or from a remote MySQL server instance. Cloned data is a physical snapshot of data stored in InnoDB that includes schemas, tables, tablespaces, and data dictionary metadata. The cloned data comprises a fully functional data directory, which permits using clone for MySQL server provisioning.

CLONE

Clone, introduced in MySQL 8.0.17, permits cloning data locally or from a remote MySQL server instance. Cloned data is a physical snapshot of data stored in InnoDB that includes schemas, tables, tablespaces, and data dictionary metadata. The cloned data comprises a fully functional data directory, which permits using clone for MySQL server provisioning.

```
 SQL > clone local data directory '/tmp/snapshot';
Query OK, 0 rows affected (5.6741 sec)
```

CLONE

Clone, introduced in MySQL 8.0.17, permits cloning data locally or from a remote MySQL server instance. Cloned data is a physical snapshot of data stored in InnoDB that includes schemas, tables, tablespaces, and data dictionary metadata. The cloned data comprises a fully functional data directory, which permits using clone for MySQL server provisioning.

```
 SQL > clone local data directory '/tmp/snapshot';
Query OK, 0 rows affected (5.6741 sec)
```

That's it ! As simple as that !

CLONE - GTID

The GTID of the snapshoted dataset can be found in `performance_schema`*:*

```
 SQL > select GTID_EXECUTED from clone_status;
+------------------------------------------------+
| GTID_EXECUTED                                  |
+------------------------------------------------+
| b00098d0-72eb-11ec-b8d2-0200170c7057:1-581783 |
+------------------------------------------------+
```

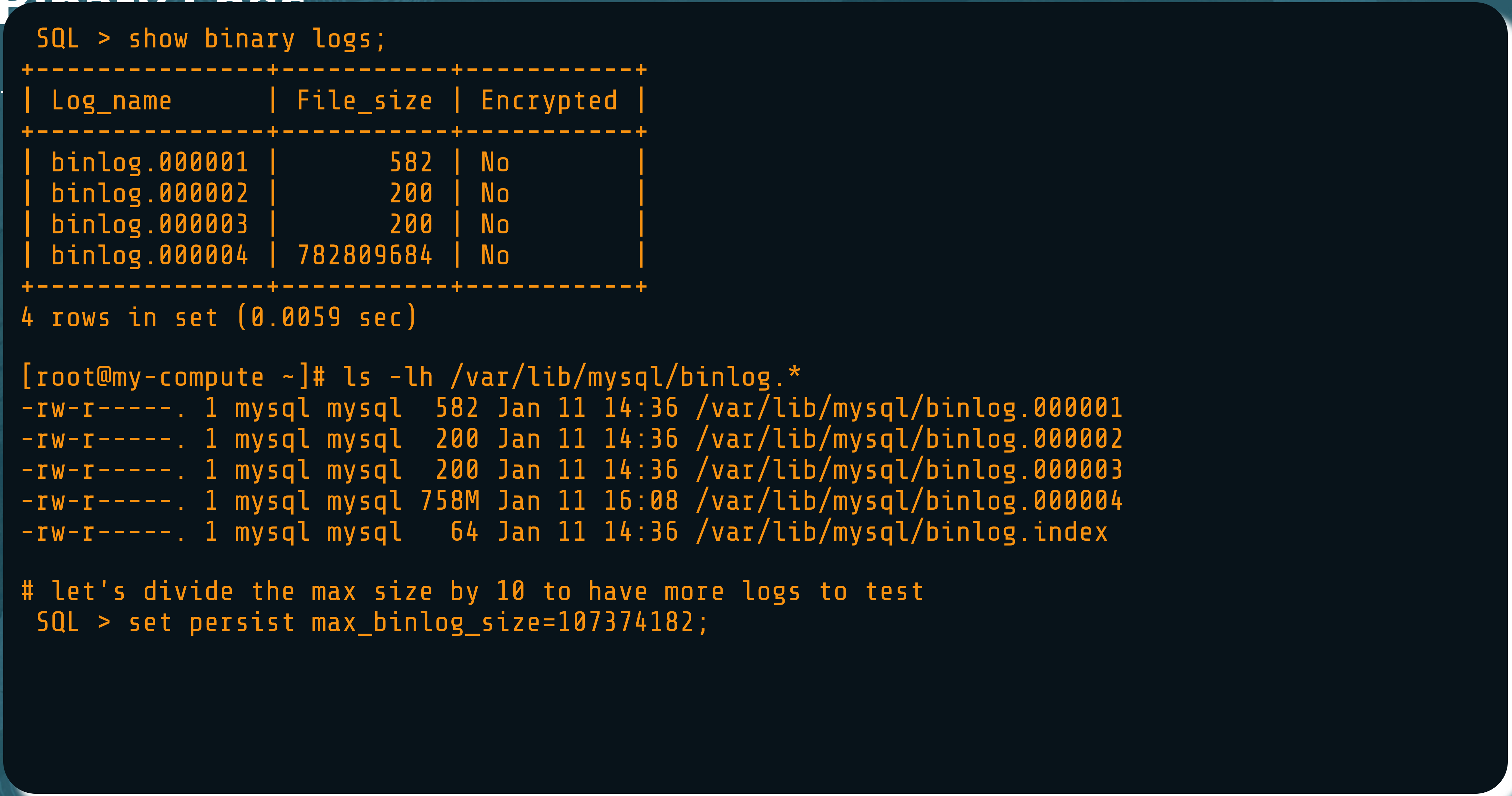
Binary logs

all the data changes are stored

Binary Logs

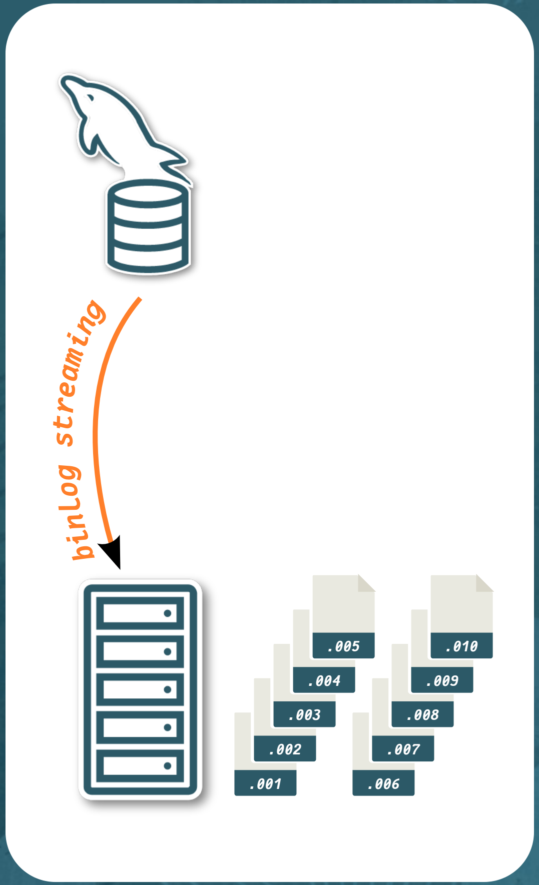
The MySQL workload is written in the binary log files:

Binary Logs



```
 SQL > show binary logs;
+---------------+-----------+-----------+
| Log_name      | File_size | Encrypted |
+---------------+-----------+-----------+
| binlog.000001 |       582 | No        |
| binlog.000002 |       200 | No        |
| binlog.000003 |       200 | No        |
| binlog.000004 | 782809684 | No        |
+---------------+-----------+-----------+
4 rows in set (0.0059 sec)

[root@my-compute ~]# ls -lh /var/lib/mysql/binlog.*
-rw-r-----. 1 mysql mysql  582 Jan 11 14:36 /var/lib/mysql/binlog.000001
-rw-r-----. 1 mysql mysql  200 Jan 11 14:36 /var/lib/mysql/binlog.000002
-rw-r-----. 1 mysql mysql  200 Jan 11 14:36 /var/lib/mysql/binlog.000003
-rw-r-----. 1 mysql mysql 758M Jan 11 16:08 /var/lib/mysql/binlog.000004
-rw-r-----. 1 mysql mysql   64 Jan 11 14:36 /var/lib/mysql/binlog.index

# let's divide the max size by 10 to have more logs to test
 SQL > set persist max_binlog_size=107374182;
```

Keeping binlogs safe



- `mysqlbinlog` *has the possibility of reading the binary logs from a live server and store them to disk using the options* `--raw --read-from-remote-server`*.*
- *we create a script to use* `mysqlbinlog` *:* `binlog_to_local.sh`
- *we use* `systemd` *to start and stop our script*

sources: https://tinyurl.com/binlogstream

Keeping binlogs safe (2)

We first need to create a dedicated user for our streaming process:

```
SQL> CREATE USER 'binlog_streamer' IDENTIFIED BY 'C0mpl1c4t3d!Passw0rd' REQUIRE SSL;
SQL> GRANT REPLICATION SLAVE ON *.* TO 'binlog_streamer';
SQL> GRANT SELECT ON performance_schema.file_instances TO 'binlog_streamer';
```

Keeping binlogs safe (2)

We first need to create a dedicated user for our streaming process:

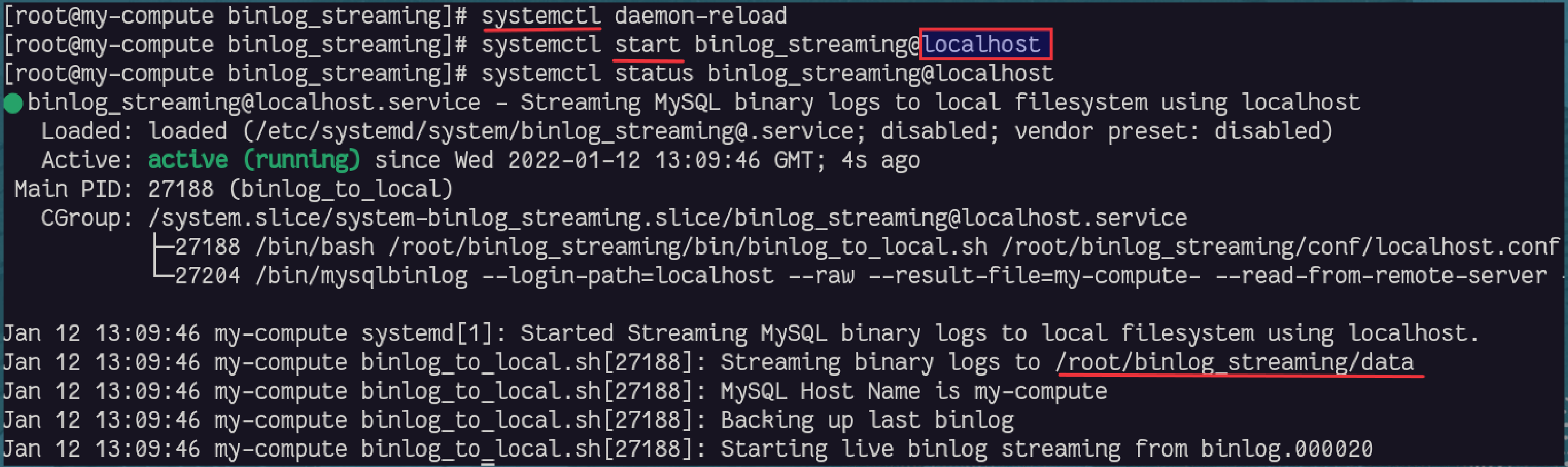
```
SQL> CREATE USER 'binlog_streamer' IDENTIFIED BY 'C0mpl1c4t3d!Passw0rd' REQUIRE SSL;
SQL> GRANT REPLICATION SLAVE ON *.* TO 'binlog_streamer';
SQL> GRANT SELECT ON performance_schema.file_instances TO 'binlog_streamer';
```

*And to avoid to store credentials in our script, let's use MySQL **Config Editor** :*

```
$ mysql_config_editor set --login-path=localhost --host=127.0.0.1 \
                          --user=binlog_streamer --password
Enter password:
```

Keeping binlogs safe (3)

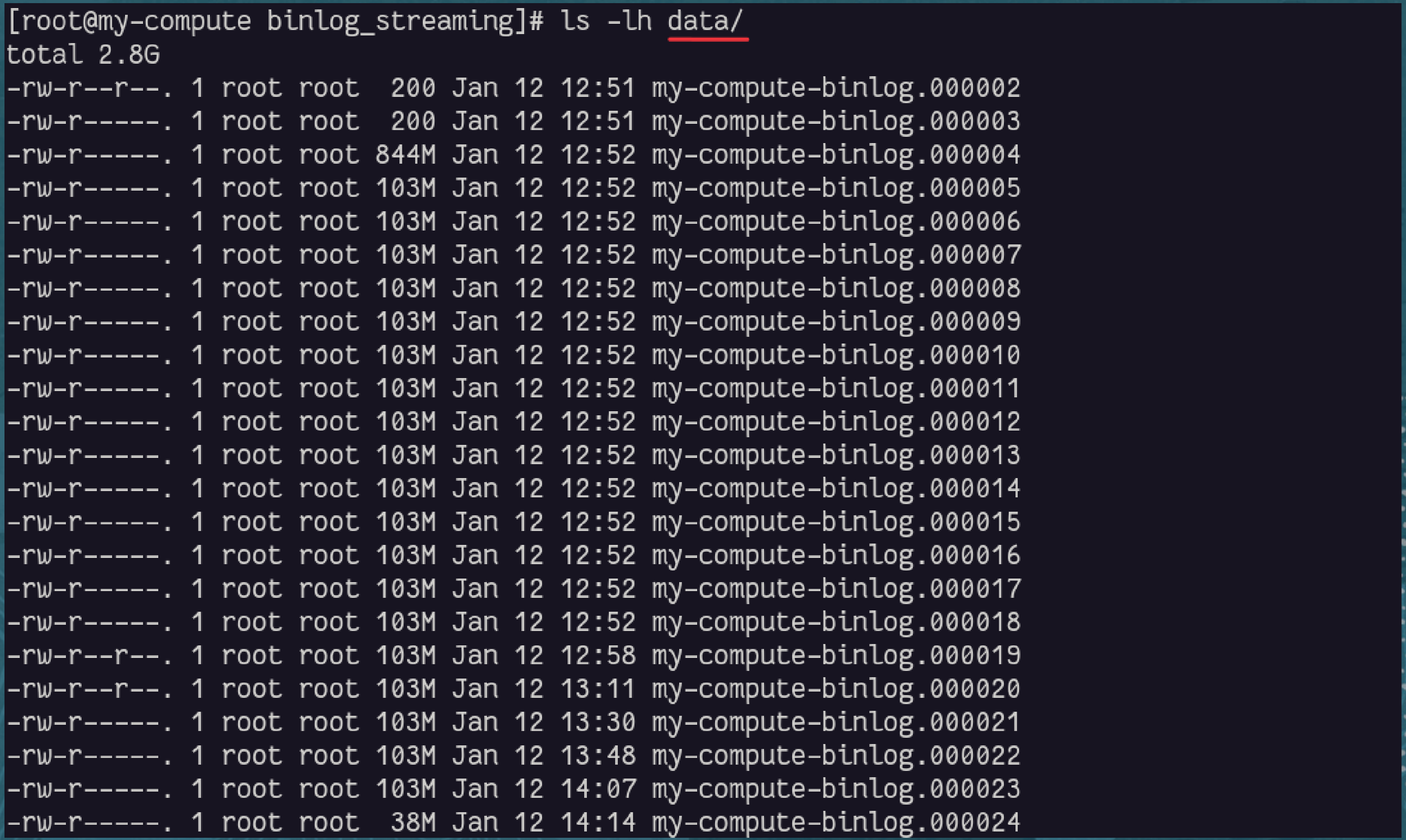
We can start the streaming using **`systemd`***:*



```
[root@my-compute binlog_streaming]# systemctl daemon-reload
[root@my-compute binlog_streaming]# systemctl start binlog_streaming@localhost
[root@my-compute binlog_streaming]# systemctl status binlog_streaming@localhost
● binlog_streaming@localhost.service - Streaming MySQL binary logs to local filesystem using localhost
   Loaded: loaded (/etc/systemd/system/binlog_streaming@.service; disabled; vendor preset: disabled)
   Active: active (running) since Wed 2022-01-12 13:09:46 GMT; 4s ago
 Main PID: 27188 (binlog_to_local)
   CGroup: /system.slice/system-binlog_streaming.slice/binlog_streaming@localhost.service
           ├─27188 /bin/bash /root/binlog_streaming/bin/binlog_to_local.sh /root/binlog_streaming/conf/localhost.conf
           └─27204 /bin/mysqlbinlog --login-path=localhost --raw --result-file=my-compute- --read-from-remote-server

Jan 12 13:09:46 my-compute systemd[1]: Started Streaming MySQL binary logs to local filesystem using localhost.
Jan 12 13:09:46 my-compute binlog_to_local.sh[27188]: Streaming binary logs to /root/binlog_streaming/data
Jan 12 13:09:46 my-compute binlog_to_local.sh[27188]: MySQL Host Name is my-compute
Jan 12 13:09:46 my-compute binlog_to_local.sh[27188]: Backing up last binlog
Jan 12 13:09:46 my-compute binlog_to_local.sh[27188]: Starting live binlog streaming from binlog.000020
```

Keeping binlogs safe (4)

The files are now also saved somewhere else (this can be another server of course):

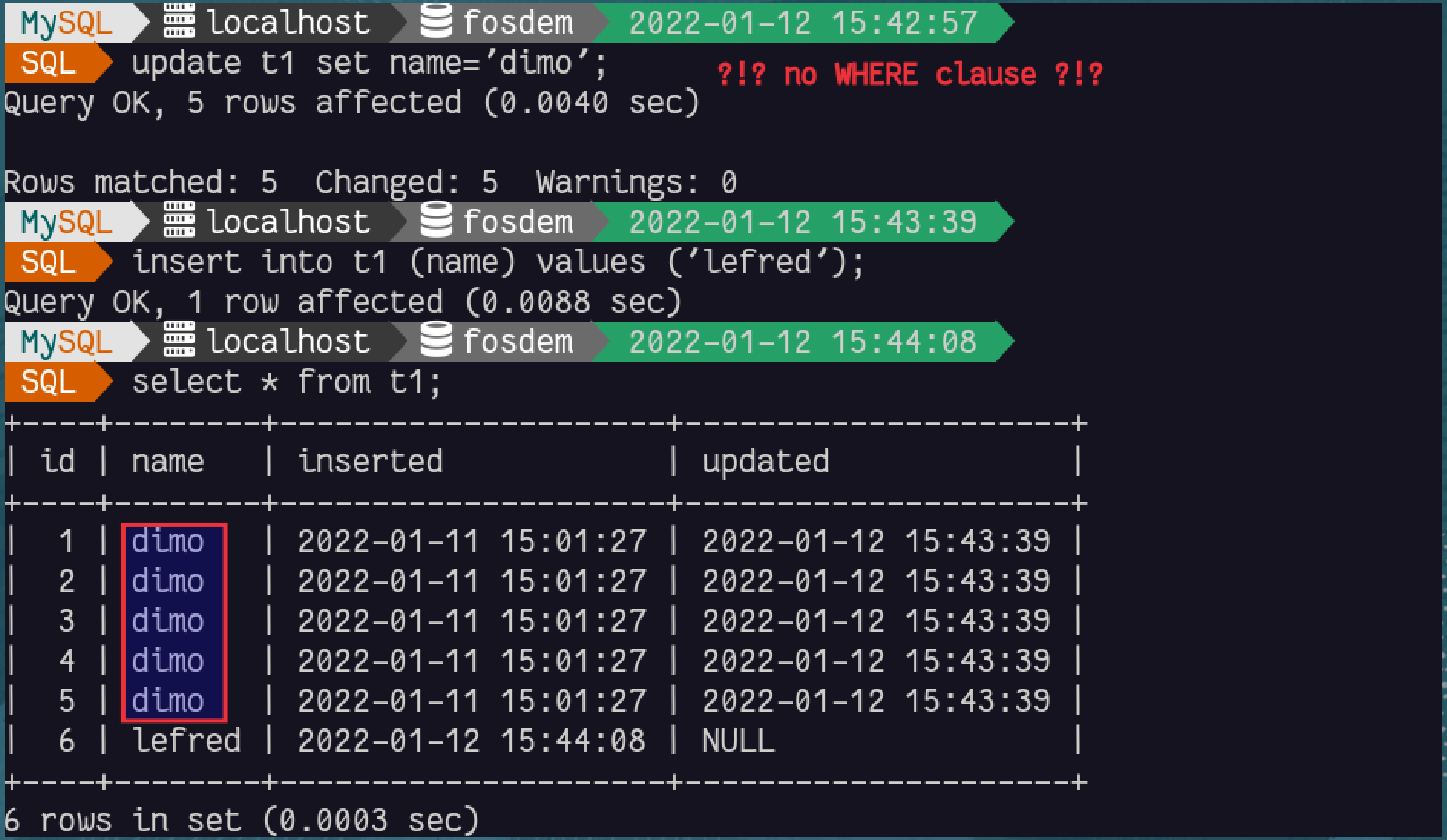


```
[root@my-compute binlog_streaming]# ls -lh data/
total 2.8G
-rw-r--r--. 1 root root  200 Jan 12 12:51 my-compute-binlog.000002
-rw-r-----. 1 root root  200 Jan 12 12:51 my-compute-binlog.000003
-rw-r-----. 1 root root 844M Jan 12 12:52 my-compute-binlog.000004
-rw-r-----. 1 root root 103M Jan 12 12:52 my-compute-binlog.000005
-rw-r-----. 1 root root 103M Jan 12 12:52 my-compute-binlog.000006
-rw-r-----. 1 root root 103M Jan 12 12:52 my-compute-binlog.000007
-rw-r-----. 1 root root 103M Jan 12 12:52 my-compute-binlog.000008
-rw-r-----. 1 root root 103M Jan 12 12:52 my-compute-binlog.000009
-rw-r-----. 1 root root 103M Jan 12 12:52 my-compute-binlog.000010
-rw-r-----. 1 root root 103M Jan 12 12:52 my-compute-binlog.000011
-rw-r-----. 1 root root 103M Jan 12 12:52 my-compute-binlog.000012
-rw-r-----. 1 root root 103M Jan 12 12:52 my-compute-binlog.000013
-rw-r-----. 1 root root 103M Jan 12 12:52 my-compute-binlog.000014
-rw-r-----. 1 root root 103M Jan 12 12:52 my-compute-binlog.000015
-rw-r-----. 1 root root 103M Jan 12 12:52 my-compute-binlog.000016
-rw-r-----. 1 root root 103M Jan 12 12:52 my-compute-binlog.000017
-rw-r-----. 1 root root 103M Jan 12 12:52 my-compute-binlog.000018
-rw-r--r--. 1 root root 103M Jan 12 12:58 my-compute-binlog.000019
-rw-r--r--. 1 root root 103M Jan 12 13:11 my-compute-binlog.000020
-rw-r-----. 1 root root 103M Jan 12 13:30 my-compute-binlog.000021
-rw-r-----. 1 root root 103M Jan 12 13:48 my-compute-binlog.000022
-rw-r-----. 1 root root 103M Jan 12 14:07 my-compute-binlog.000023
-rw-r-----. 1 root root  38M Jan 12 14:14 my-compute-binlog.000024
```

Point-in-Time Recovery

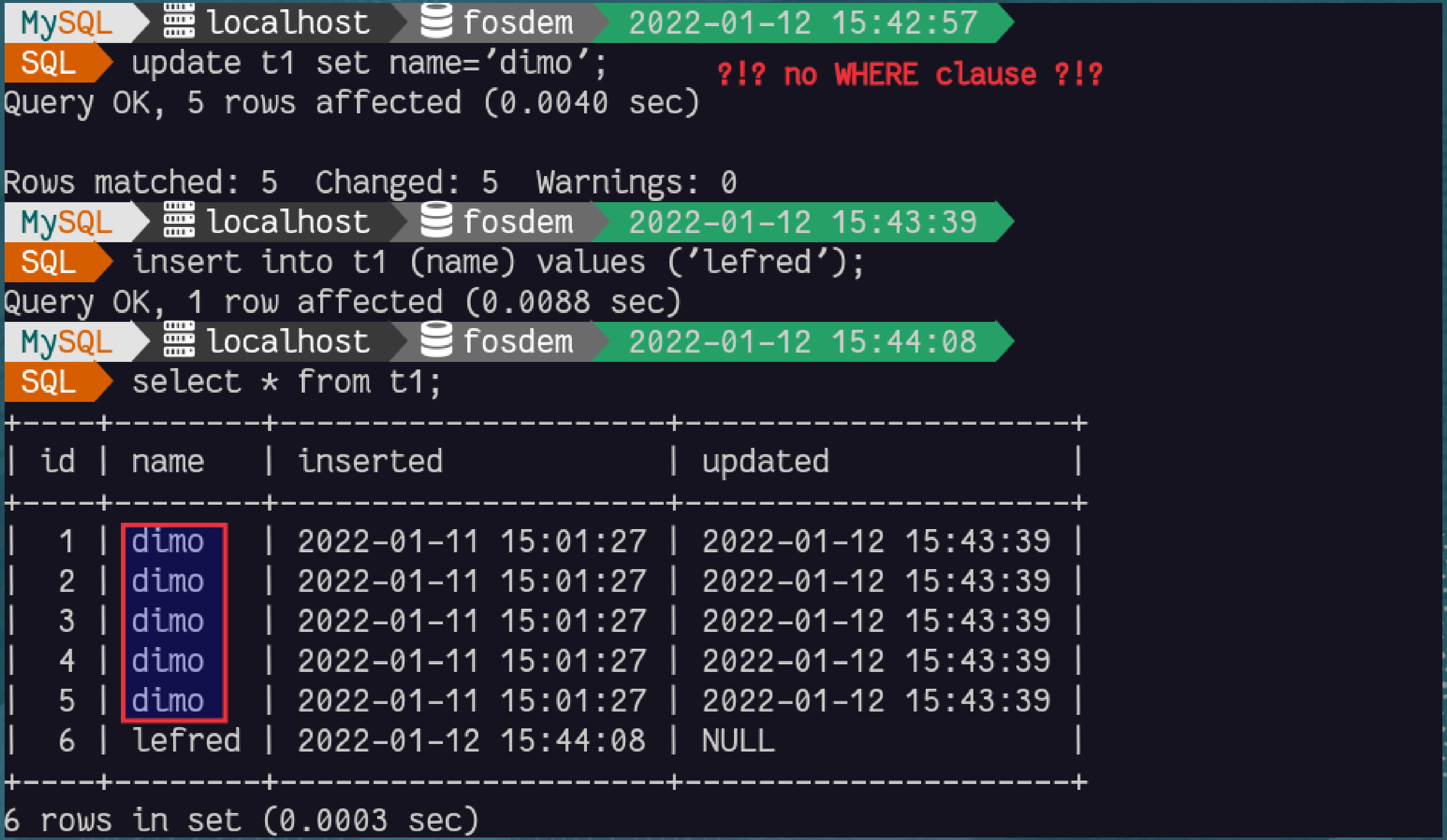
examples

Something we would like to avoid…



```
MySQL  localhost  fosdem  2022-01-12 15:42:57
SQL  update t1 set name='dimo';       ?!? no WHERE clause ?!?
Query OK, 5 rows affected (0.0040 sec)

Rows matched: 5  Changed: 5  Warnings: 0
MySQL  localhost  fosdem  2022-01-12 15:43:39
SQL  insert into t1 (name) values ('lefred');
Query OK, 1 row affected (0.0088 sec)
MySQL  localhost  fosdem  2022-01-12 15:44:08
SQL  select * from t1;
+----+--------+---------------------+---------------------+
| id | name   | inserted            | updated             |
+----+--------+---------------------+---------------------+
|  1 | dimo   | 2022-01-11 15:01:27 | 2022-01-12 15:43:39 |
|  2 | dimo   | 2022-01-11 15:01:27 | 2022-01-12 15:43:39 |
|  3 | dimo   | 2022-01-11 15:01:27 | 2022-01-12 15:43:39 |
|  4 | dimo   | 2022-01-11 15:01:27 | 2022-01-12 15:43:39 |
|  5 | dimo   | 2022-01-11 15:01:27 | 2022-01-12 15:43:39 |
|  6 | lefred | 2022-01-12 15:44:08 | NULL                |
+----+--------+---------------------+---------------------+
6 rows in set (0.0003 sec)
```

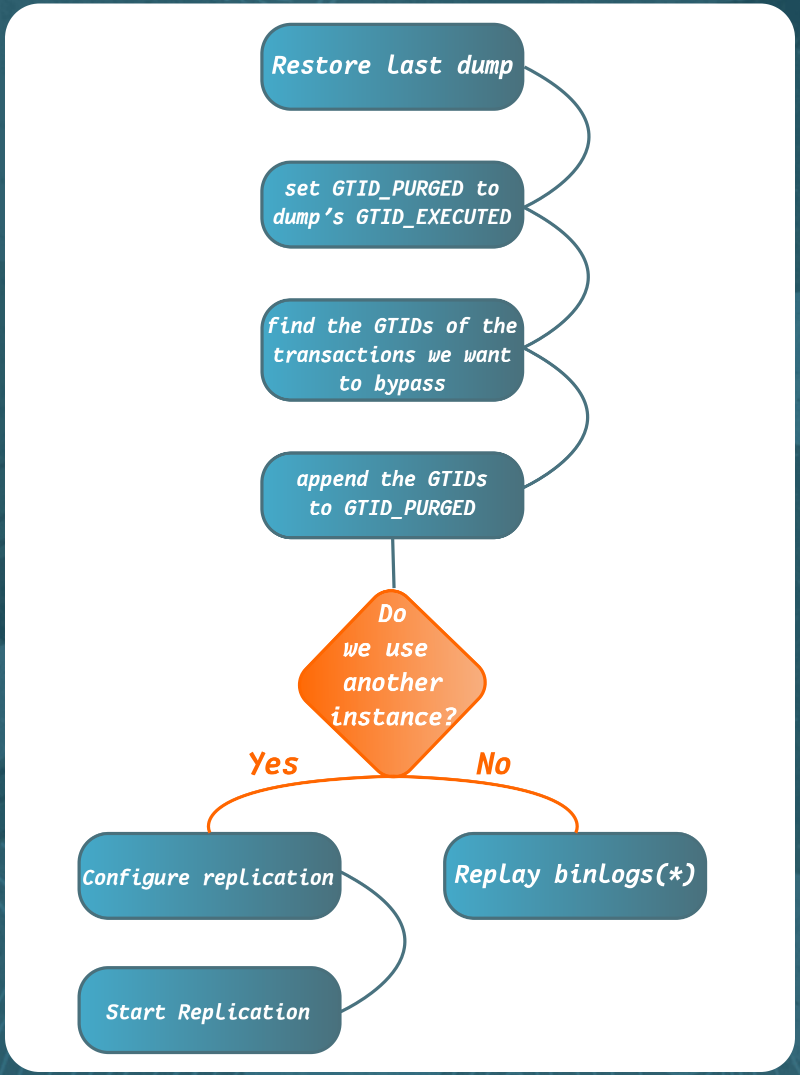
Something we would like to avoid…



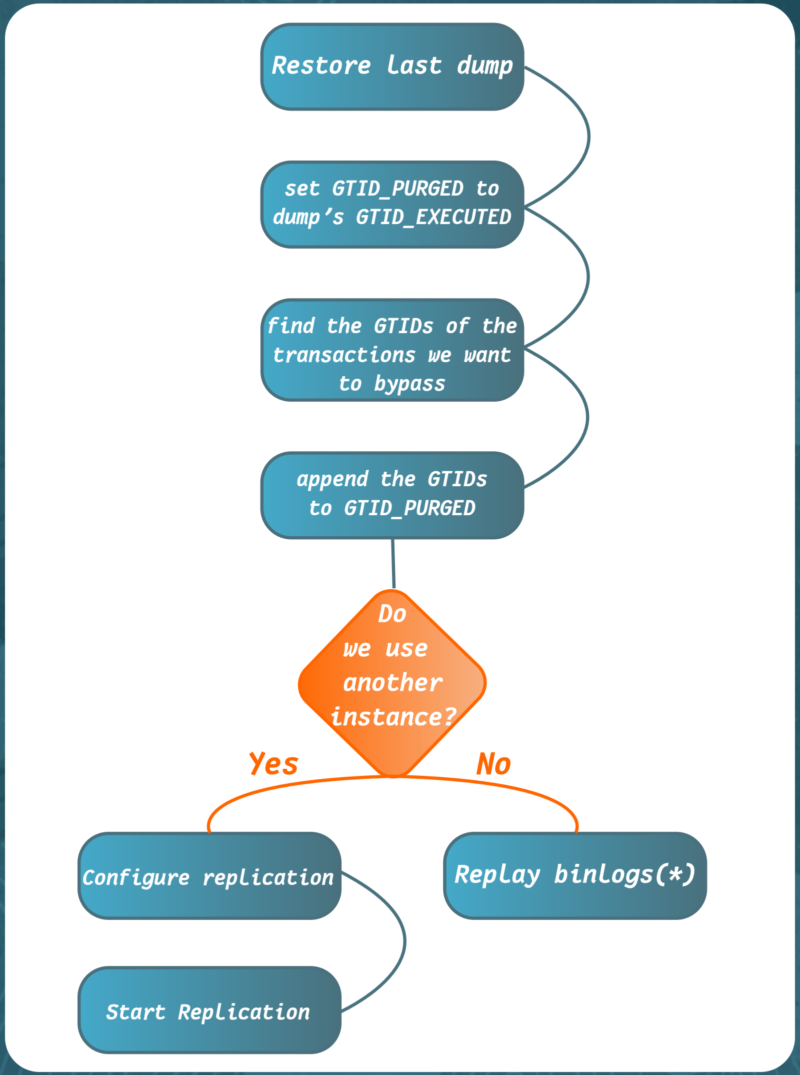
```
MySQL  localhost  fosdem  2022-01-12 15:42:57
SQL  update t1 set name='dimo';
Query OK, 5 rows affected (0.0040 sec)

Rows matched: 5  Changed: 5  Warnings: 0
MySQL  localhost  fosdem  2022-01-12 15:43:39
SQL  insert into t1 (name) values ('lefred');
Query OK, 1 row affected (0.0088 sec)
MySQL  localhost  fosdem  2022-01-12 15:44:08
SQL  select * from t1;
+----+--------+---------------------+---------------------+
| id | name   | inserted            | updated             |
+----+--------+---------------------+---------------------+
|  1 | dimo   | 2022-01-11 15:01:27 | 2022-01-12 15:43:39 |
|  2 | dimo   | 2022-01-11 15:01:27 | 2022-01-12 15:43:39 |
|  3 | dimo   | 2022-01-11 15:01:27 | 2022-01-12 15:43:39 |
|  4 | dimo   | 2022-01-11 15:01:27 | 2022-01-12 15:43:39 |
|  5 | dimo   | 2022-01-11 15:01:27 | 2022-01-12 15:43:39 |
|  6 | lefred | 2022-01-12 15:44:08 | NULL                |
+----+--------+---------------------+---------------------+
6 rows in set (0.0003 sec)
```

?!? no WHERE clause ?!?

The Action Plan


Restore last dump
set GTID_PURGED to dump’s GTID_EXECUTED
find the GTIDs of the transactions we want to bypass
append the GTIDs to GTID_PURGED
Do we use another instance?
Yes
No
Configure replication
Start Replication
Replay binlogs(*)

The Action Plan


Restore last dump
set GTID_PURGED to dump's GTID_EXECUTED
find the GTIDs of the transactions we want to bypass
append the GTIDs to GTID_PURGED
Do we use another instance?
Yes
No
Configure replication
Start Replication
Replay binlogs(*)

I chose to perform point-in-time recovery on the ***same*** *machine to show how we can accelerate the process.*

Before we start

Some actions are necessary before we start the point-in-time recovery process:

- *if you plan to do point-in-time recovery on the same instance, you need to stop the application (*`sysbench` *in our case)*
- *we check the last* `GTID_EXECUTED`
- *we do* `select count(*)` *on the* `sysbench` *tables just to have an estimation we recovered as expected.*
- *stop the binlog streaming process*

Before we start (2)

```
 SQL > select @@GTID_EXECUTED;
+-------------------------------------------------+
| @@GTID_EXECUTED                                 |
+-------------------------------------------------+
| b00098d0-72eb-11ec-b8d2-0200170c7057:1-5854318 |
+-------------------------------------------------+
```

```
 SQL > select  (select count(*) from sbtest.sbtest1) t1,
                ... ,
               (select count(*) from sbtest.sbtest8) t8;
+--------+--------+--------+--------+--------+--------+--------+--------+
| t1     | t2     | t3     | t4     | t5     | t6     | t7     | t8     |
+--------+--------+--------+--------+--------+--------+--------+--------+
| 667831 | 670327 | 669287 | 668361 | 668443 | 668736 | 670188 | 669557 |
+--------+--------+--------+--------+--------+--------+--------+--------+
```

```
$ sudo systemctl stop binlog_streaming@localhost.service
```

Restore Last Dump

We have again serveral options:

- *restore the logical dump made with MySQL Shell*
- *restore the snapshot made with CLONE.*

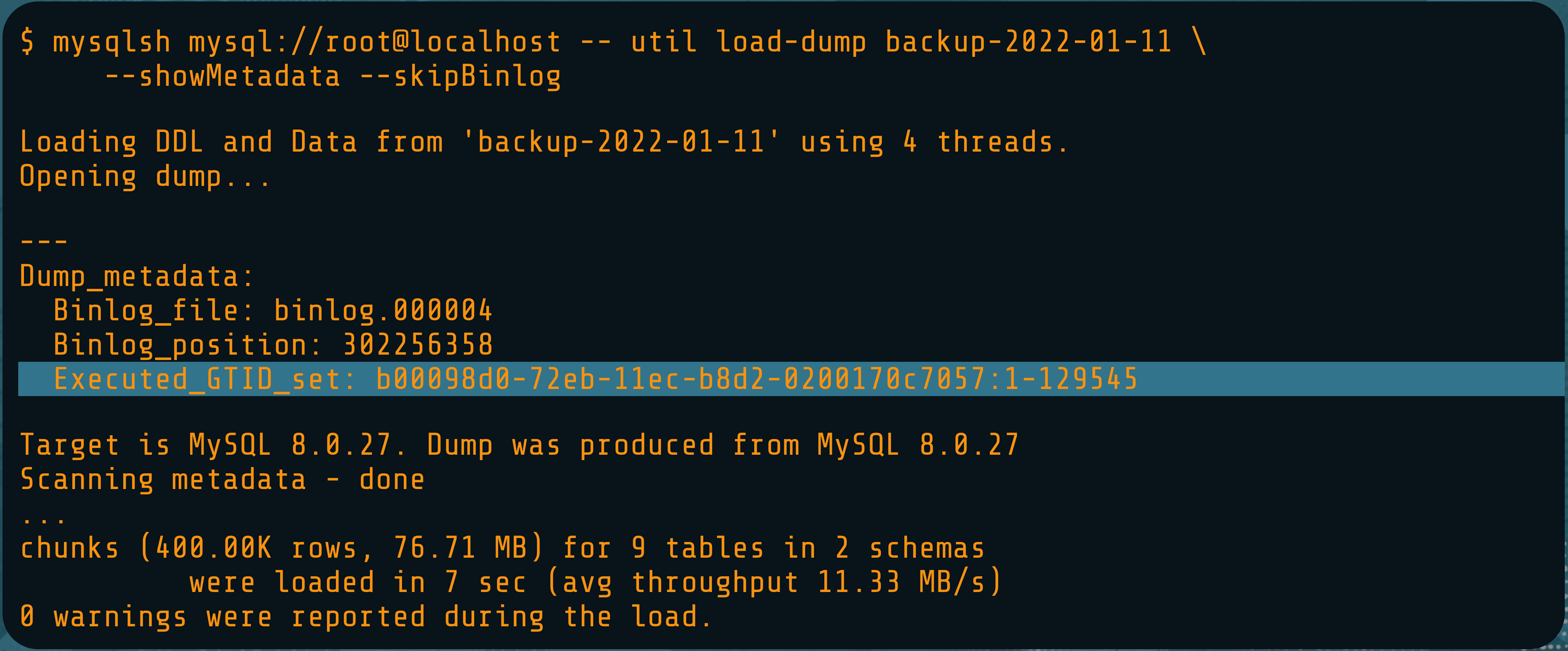
Restore Last Dump - MySQL Shell Utility

To restore a dump made with MySQL Shell Dump Utility, we need a MySQL server running.

We need to remove all non system schemas:

```
 SQL > drop schema fosdem;
Query OK, 1 row affected (0.0225 sec)

 SQL > drop schema sbtest;
Query OK, 8 rows affected (0.2117 sec)
```

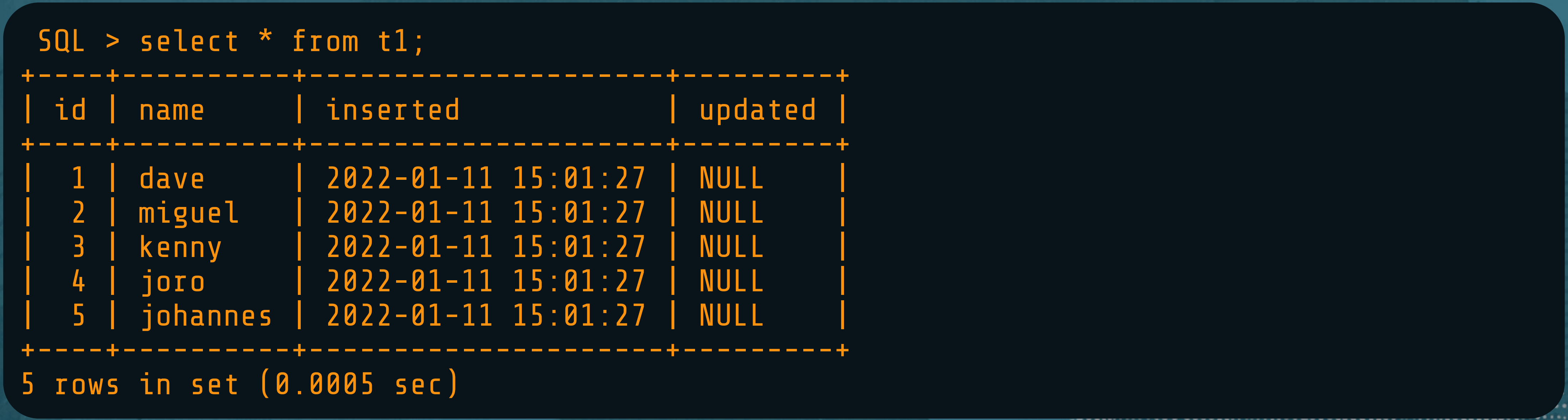
Restore Last Dump - MySQL Shell Utility (2)



```
$ mysqlsh mysql://root@localhost -- util load-dump backup-2022-01-11 \
     --showMetadata --skipBinlog

Loading DDL and Data from 'backup-2022-01-11' using 4 threads.
Opening dump...

---
Dump_metadata:
  Binlog_file: binlog.000004
  Binlog_position: 302256358
  Executed_GTID_set: b00098d0-72eb-11ec-b8d2-0200170c7057:1-129545

Target is MySQL 8.0.27. Dump was produced from MySQL 8.0.27
Scanning metadata - done
...
chunks (400.00K rows, 76.71 MB) for 9 tables in 2 schemas
          were loaded in 7 sec (avg throughput 11.33 MB/s)
0 warnings were reported during the load.
```

Restore Last Dump - MySQL Shell Utility (3)

We can already check if our table looks like what it was before the dump:



```
 SQL > select * from t1;
+----+----------+---------------------+---------+
| id | name     | inserted            | updated |
+----+----------+---------------------+---------+
|  1 | dave     | 2022-01-11 15:01:27 | NULL    |
|  2 | miguel   | 2022-01-11 15:01:27 | NULL    |
|  3 | kenny    | 2022-01-11 15:01:27 | NULL    |
|  4 | joro     | 2022-01-11 15:01:27 | NULL    |
|  5 | johannes | 2022-01-11 15:01:27 | NULL    |
+----+----------+---------------------+---------+
5 rows in set (0.0005 sec)
```

Restore Last Dump - MySQL Shell Utility (4)

We still need to set back the GTIDs:

```
 SQL > select @@gtid_purged, @@gtid_executed;
+---------------+------------------------------------------------+
| @@gtid_purged | @@gtid_executed                                |
+---------------+------------------------------------------------+
|               | b00098d0-72eb-11ec-b8d2-0200170c7057:1-5854320 |
+---------------+------------------------------------------------+
```

Restore Last Dump - MySQL Shell Utility (4)

We still need to set back the GTIDs:

```
 SQL > select @@gtid_purged, @@gtid_executed;
+---------------+------------------------------------------------+
| @@gtid_purged | @@gtid_executed                                |
+---------------+------------------------------------------------+
|               | b00098d0-72eb-11ec-b8d2-0200170c7057:1-5854320 |
+---------------+------------------------------------------------+
```

```
 SQL > reset master;
 SQL > set global gtid_purged='b00098d0-72eb-11ec-b8d2-0200170c7057:1-129545';
 SQL > select @@gtid_purged, @@gtid_executed;
+---------------------------------+--------------------------------------+
| @@gtid_purged                   | @@gtid_executed                      |
+---------------------------------+--------------------------------------+
| b00098d0-72eb-11ec-...:1-129545 | b00098d0-72eb-11ec-b8d2-...:1-129545 |
+---------------------------------+--------------------------------------+
```

Restore Last Dump - CLONE

As the plan is to retore the snapshot on the same server, we need first to save 2 imporant files from MySQL's data directory:

- `auto.cnf`*: containing the server-uuid*
- `mysqld-auto.cnf`*: containing all configuration changes done using* `SET PERSIST`
- *additionnaly if you have your own dedicated keys in the datadir, you should also save them*

Restore Last Dump - CLONE (2)

Let's starti by saving the required files:

```
$ sudo cp /var/lib/mysql/auto.cnf  snapshot
$ sudo cp /var/lib/mysql/mysqld-auto.cnf  snapshot
```

Restore Last Dump - CLONE (2)

Let's starti by saving the required files:

```
$ sudo cp /var/lib/mysql/auto.cnf  snapshot
$ sudo cp /var/lib/mysql/mysqld-auto.cnf  snapshot
```

And now we stop MySQL and empty the datadir:

```
$ sudo systemctl stop mysqld
$ sudo rm -rf /var/lib/mysql/*
```

Restore Last Dump - CLONE (2)

Let's starti by saving the required files:

```
$ sudo cp /var/lib/mysql/auto.cnf  snapshot
$ sudo cp /var/lib/mysql/mysqld-auto.cnf  snapshot
```

And now we stop MySQL and empty the datadir:

```
$ sudo systemctl stop mysqld
$ sudo rm -rf /var/lib/mysql/*
```

Let's copy back the files from the snapshot and start MySQL:

```
$ sudo cp -r snapshot/* /var/lib/mysql
$ sudo chown -R mysql. /var/lib/mysql
$ sudo systemctl start mysqld
```

Restore Last Dump - CLONE (3)

We can see that the GTIDs are already in place:

```
 SQL > select @@gtid_purged, @@gtid_executed;
+--------------------------------------+--------------------------------------+
| @@gtid_purged                        | @@gtid_executed                      |
+--------------------------------------+--------------------------------------+
| b00098d0-72eb-11ec-b8d2-...:1-581783 | b00098d0-72eb-11ec-b8d2-...:1-581783 |
+--------------------------------------+--------------------------------------+
```

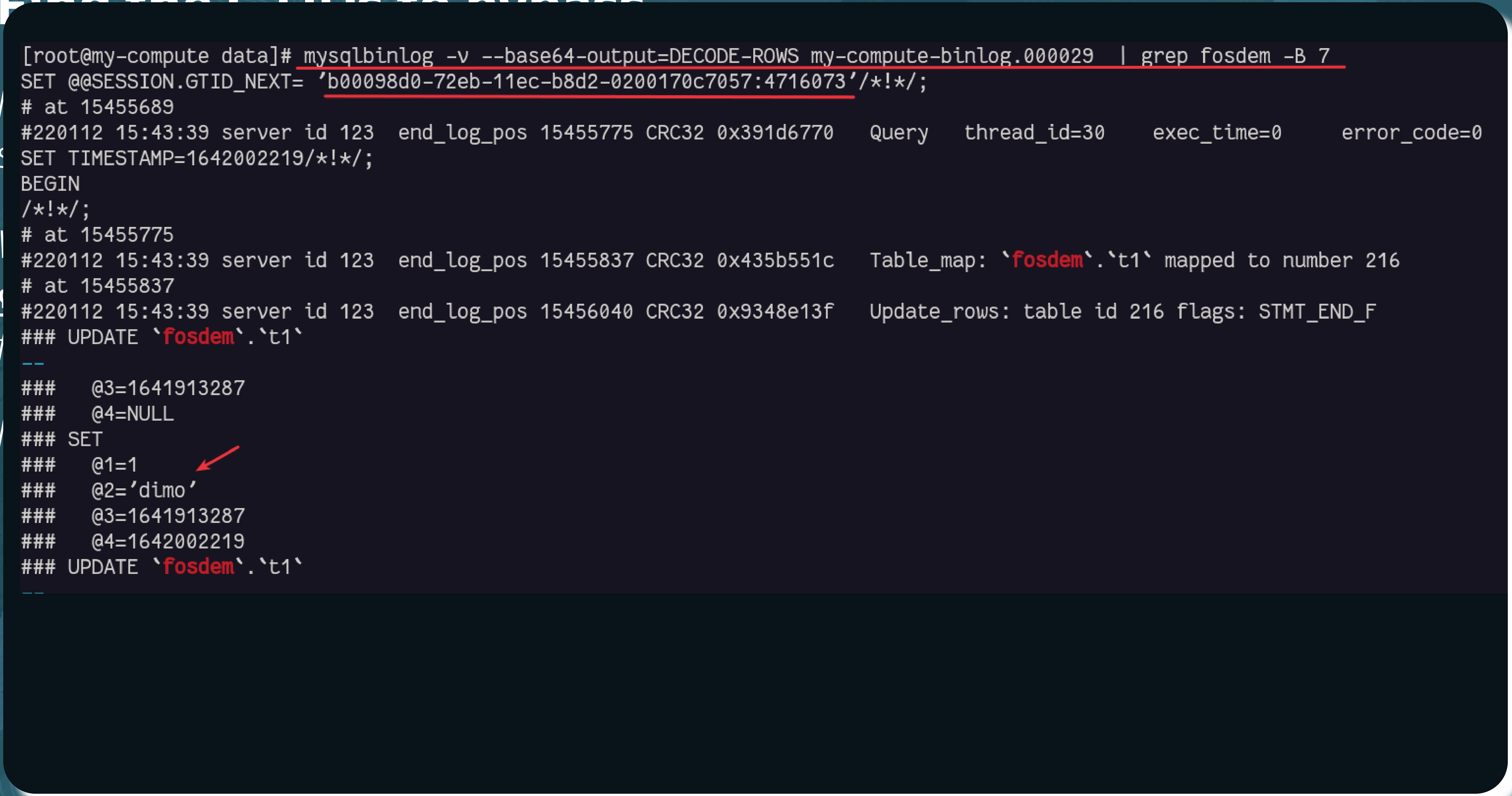
Find the GTIDs to bypass

Now on the binary logs we have streamed, we need to find the transaction(s) we want to skip.

We use `mysqlbinlog -v --base64-output=DECODE-ROWS <binlog file>` *with* `grep` *to find the right file. The timestamp on the file can of course help to dentify the right file.*

I found that the file is `my-compute-binlog.000029`*.*

Find the GTIDs to bypass



```
[root@my-compute data]# mysqlbinlog -v --base64-output=DECODE-ROWS my-compute-binlog.000029  | grep fosdem -B 7
SET @@SESSION.GTID_NEXT= 'b00098d0-72eb-11ec-b8d2-0200170c7057:4716073'/*!*/;
# at 15455689
#220112 15:43:39 server id 123  end_log_pos 15455775 CRC32 0x391d6770   Query   thread_id=30    exec_time=0     error_code=0
SET TIMESTAMP=1642002219/*!*/;
BEGIN
/*!*/;
# at 15455775
#220112 15:43:39 server id 123  end_log_pos 15455837 CRC32 0x435b551c   Table_map: `fosdem`.`t1` mapped to number 216
# at 15455837
#220112 15:43:39 server id 123  end_log_pos 15456040 CRC32 0x9348e13f   Update_rows: table id 216 flags: STMT_END_F
### UPDATE `fosdem`.`t1`
--
###   @3=1641913287
###   @4=NULL
### SET
###   @1=1
###   @2='dimo'
###   @3=1641913287
###   @4=1642002219
### UPDATE `fosdem`.`t1`
--
```

Skip the GTIDs

It's time now to tell MySQL which GTIDs we want to avoid (only one in our example).

To do so, we will append to the `GTID_PURGED` *the GTIDs we want to skip:*

```
 SQL > SET @@GLOBAL.gtid_purged = '+b00098d0-72eb-11ec-b8d2-0200170c7057:4716073';
Query OK, 0 rows affected (0.0045 sec)

 SQL    select @@gtid_purged, @@gtid_executed\G
 *************************** 1. row ***************************
  @@gtid_purged: b00098d0-72eb-11ec-b8d2-0200170c7057:1-581783:4716073
@@gtid_executed: b00098d0-72eb-11ec-b8d2-0200170c7057:1-581783:4716073
```

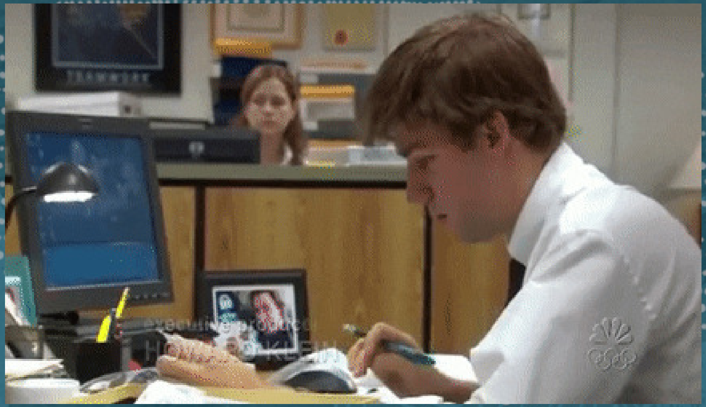
Replay the Binary Logs

Now we could replay the binary logs one by one to our MySQL server... but that can lead to a very long operation as `mysqlbinlog` *is single-threaded.*

Unfortunately, on a Cloud manage instance, this is the only feasible method:

```
$ mysqlbinlog my-compute-binlog.000002 | mysql
```

And repeat this for all binary logs...



Replay the Binary Logs… like a Rockstar !

We will let believe to MySQL that those streamed binary logs are relay logs !

Therefore, MySQL will be able to ingest them in parallel very quickly !

Replay the Binary Logs... like a Rockstar !

We will let believe to MySQL that those streamed binary logs are relay logs !

Therefore, MySQL will be able to ingest them in parallel very quickly !

```
 SQL > select @@relay_log;
+-----------------------+
| @@relay_log           |
+-----------------------+
| my-compute-relay-bin  |
+-----------------------+
```

Replay the Binary Logs... like a Rockstar ! (2)

Let's copy the files:

```
$ cd /mnt/binlog_streaming/data
$ for i in `ls *`; do
    sudo cp $i /var/lib/mysql/my-compute-relay-bin.${i#*.}
  done
$ chown mysql. /var/lib/mysql/my-compute-relay-bin.*
```

And of course we need to create the relay index file too:

```
$ cd /var/lib/mysql
$ sudo ls ./my-compute-relay-bin.* > my-compute-relay-bin.index
$ sudo chown mysql. my-compute-relay-bin.index
```

Replay the Binary Logs... like a Rockstar ! (3)

Let's verify that we can ingest to relay logs in parallel:

```
 SQL > select @@replica_parallel_type, @@replica_parallel_workers;
+--------------------------+-----------------------------+
| @@replica_parallel_type | @@replica_parallel_workers |
+--------------------------+-----------------------------+
| LOGICAL_CLOCK            |                           4 |
+--------------------------+-----------------------------+
```

This is enough on my system but don't hesitate to increase the threads if you have CPU power.

> *If you can afford a MySQL restart before and after pitr, it might be good to set `log_replica_updates` to `0`.*

Replay the Binary Logs… like a Rockstar ! (4)

And now… let's start !

```
 SQL > SET GLOBAL server_id = 99;
Query OK, 0 rows affected (0.0003 sec)

 SQL> SET GLOBAL binlog_transaction_dependency_tracking='writeset';
Query OK, 0 rows affected (0.0002 sec)

 SQL > CHANGE REPLICATION SOURCE
       TO RELAY_LOG_FILE='my-compute-relay-bin.000002',
       RELAY_LOG_POS=4, SOURCE_HOST='dummy';
Query OK, 0 rows affected (0.1464 sec)

 SQL > START REPLICA SQL_THREAD;
Query OK, 0 rows affected (0.0144 sec)
```

Replay the Binary Logs… like a Rockstar ! (5)

You can verify the progress in `performance_schema` *in tables* `replication_applier_status_by_coordinator` *and* `replication_applier_status_by_worker`*:*

```
 SQL > SELECT LAST_APPLIED_TRANSACTION, APPLYING_TRANSACTION,
              APPLYING_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP
       FROM performance_schema.replication_applier_status_by_worker\G
 *************************** 1. row ***************************
                LAST_APPLIED_TRANSACTION: b00098d0-72eb-11ec-b8d2-0200170c7057:607684
                    APPLYING_TRANSACTION: b00098d0-72eb-11ec-b8d2-0200170c7057:607685
APPLYING_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP: 2022-01-11 16:41:52.849261
 *************************** 2. row ***************************
                LAST_APPLIED_TRANSACTION: b00098d0-72eb-11ec-b8d2-0200170c7057:607369
                    APPLYING_TRANSACTION:
...
```

Test

Let's now verify...

```
 SQL > select @@gtid_purged, @@gtid_executed;
+-----------------------------------------------+----------------------------------+
| @@gtid_purged                                 | @@gtid_executed                  |
+-----------------------------------------------+----------------------------------+
| b00098d0-72eb-11ec-...:1-581783:735903-5854318 | b00098d0-72eb-11ec-...:1-5854318 |
+-----------------------------------------------+----------------------------------+
```

Test

Let's now verify…

```
 SQL > select @@gtid_purged, @@gtid_executed;
+----------------------------------------------+-----------------------------------+
| @@gtid_purged                                | @@gtid_executed                   |
+----------------------------------------------+-----------------------------------+
| b00098d0-72eb-11ec-...:1-581783:735903-5854318 | b00098d0-72eb-11ec-...:1-5854318 |
+----------------------------------------------+-----------------------------------+
```

```
 SQL > select  (select count(*) from sbtest.sbtest1) t1,
                ... ,
               (select count(*) from sbtest.sbtest8) t8;
+--------+--------+--------+--------+--------+--------+--------+--------+
| t1     | t2     | t3     | t4     | t5     | t6     | t7     | t8     |
+--------+--------+--------+--------+--------+--------+--------+--------+
| 667831 | 670327 | 669287 | 668361 | 668443 | 668736 | 670188 | 669557 |
+--------+--------+--------+--------+--------+--------+--------+--------+
```

Test (2)

And finally:

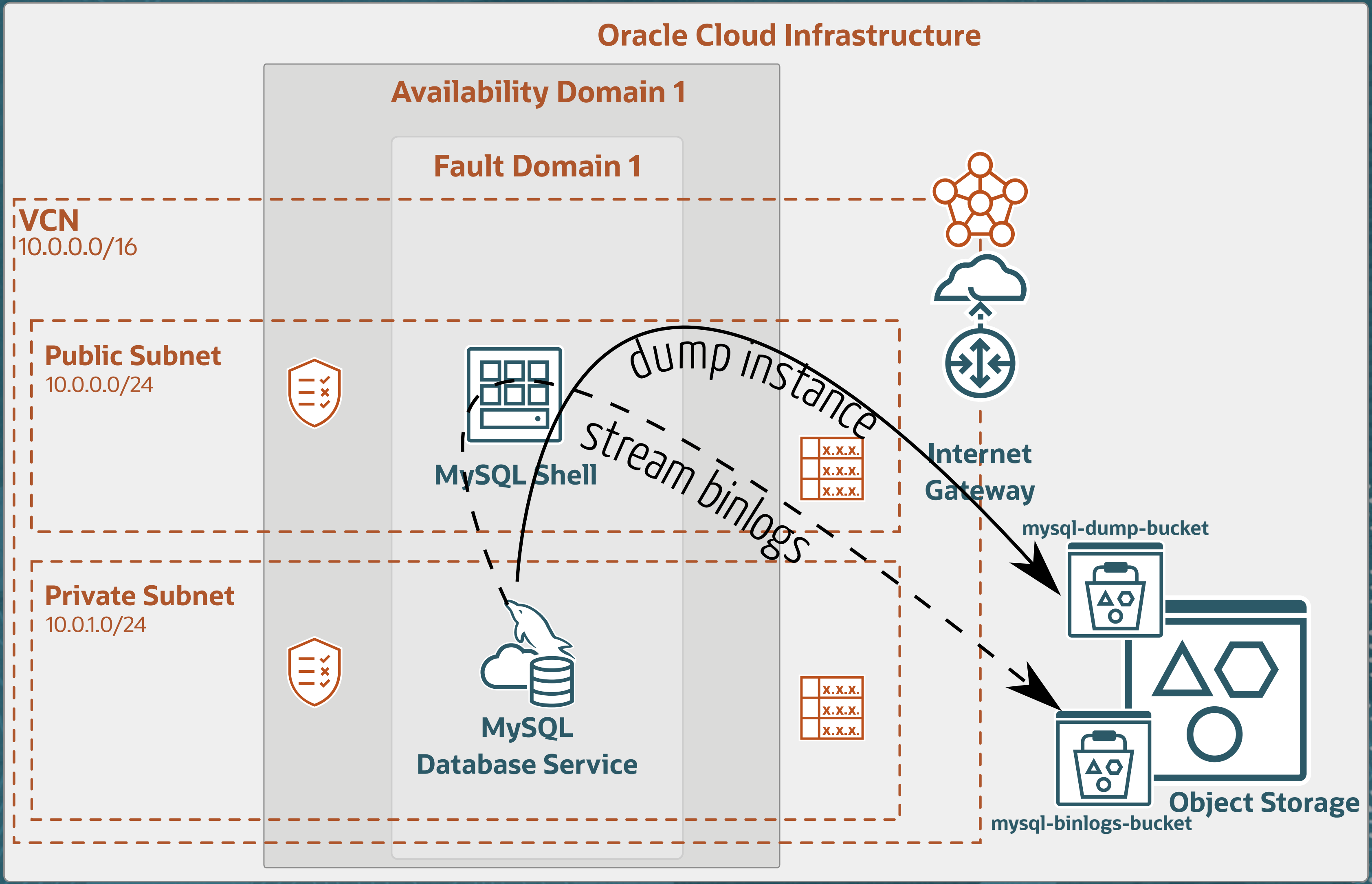
```
SQL > select * from fosdem.t1;
+----+----------+---------------------+---------+
| id | name     | inserted            | updated |
+----+----------+---------------------+---------+
|  1 | dave     | 2022-01-11 15:01:27 | NULL    |
|  2 | miguel   | 2022-01-11 15:01:27 | NULL    |
|  3 | kenny    | 2022-01-11 15:01:27 | NULL    |
|  4 | joro     | 2022-01-11 15:01:27 | NULL    |
|  5 | johannes | 2022-01-11 15:01:27 | NULL    |
|  6 | lefred   | 2022-01-12 15:44:08 | NULL    |
+----+----------+---------------------+---------+
6 rows in set (0.0006 sec)
```

Don't forget to put back the initial value of `server_id`

And in the cloud ?

Setting up your strategy in OCI

Strategy in OCI with MDS



Strategy in OCI with MDS (2)

- *Backups/snapshots are managed by the MySQL Team*
- *Binary logs are purged every hour by default*
- *You need to stream your Binary logs to Object Storage using a dedicated compute instance*
- *You can also perform logical dumps to Object Storage (not mandatory)*

Strategy in OCI with MDS (3)

More details:

- *https://lefred.be/content/point-in-time-recovery-in-oci-mds-with-object-storage-part-1/*
- *https://lefred.be/content/point-in-time-recovery-in-oci-mds-with-object-storage-part-2/*

Questions ?