Elettra
Sincrotrone
Trieste

TANGO

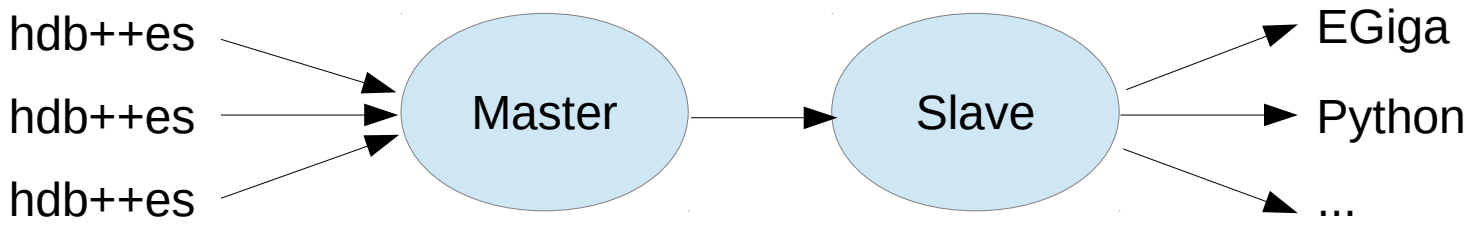
HDB++ InnoDB partitioning @Elettra

Graziano Scalamera

HDB++ @ Elettra

Fermi

- MySQL version 5.7.20
- MySQL engine InnoDB
- Time based partitioning: all tables partitioned once per year except att_scalar_devdouble_ro partitioned every 2 months
- Master – Slave replication:



HDB++ @ Elettra

InnoDB schema

```
CREATE TABLE IF NOT EXISTS att_scalar_devuchar_ro
(
   att_conf_id INT UNSIGNED NOT NULL,
   data_time DATETIME(6) NOT NULL,
   recv_time DATETIME(6) NOT NULL,
   value_r TINYINT UNSIGNED DEFAULT NULL,
   quality TINYINT(1) DEFAULT NULL,
   att_error_desc_id INT UNSIGNED NULL DEFAULT NULL,
 PRIMARY KEY(att_conf_id, data_time)
) ENGINE=InnoDB
PARTITION BY RANGE COLUMNS(data_time) (
      PARTITION p000 VALUES LESS THAN ('2015-01-01'),
      PARTITION p2015 VALUES LESS THAN ('2016-01-01'),
      PARTITION p2016 VALUES LESS THAN ('2017-01-01'),
      PARTITION p2017 VALUES LESS THAN ('2018-01-01'),
      PARTITION p2018 VALUES LESS THAN ('2019-01-01'),
      PARTITION p2019 VALUES LESS THAN ('2020-01-01'),
      PARTITION future      VALUES LESS THAN MAXVALUE
);
```

HDB++ @ Elettra

InnoDB schema

```
CREATE TABLE IF NOT EXISTS att_scalar_devuchar_ro
(
   att_conf_id INT UNSIGNED NOT NULL,
   data_time DATETIME(6) NOT NULL,
   recv_time DATETIME(6) NOT NULL,
   value_r TINYINT UNSIGNED DEFAULT NULL,
   quality TINYINT(1) DEFAULT NULL,
   att_error_desc_id INT UNSIGNED NULL DEFAULT NULL,
 PRIMARY KEY(att_conf_id, data_time)
) ENGINE=InnoDB
PARTITION BY RANGE COLUMNS(data_time) (
      PARTITION p000 VALUES LESS THAN ('2015-01-01'),
      PARTITION p2015 VALUES LESS THAN ('2016-01-01'),
      PARTITION p2016 VALUES LESS THAN ('2017-01-01'),
      PARTITION p2017 VALUES LESS THAN ('2018-01-01'),
      PARTITION p2018 VALUES LESS THAN ('2019-01-01'),
      PARTITION p2019 VALUES LESS THAN ('2020-01-01'),
      PARTITION future      VALUES LESS THAN MAXVALUE
);
```

Need to use DATETIME and not TIMESTAMP because of issues with range partitioning

HDB++ @ Elettra

InnoDB schema

```
CREATE TABLE IF NOT EXISTS att_scalar_devuchar_ro
(
   att_conf_id INT UNSIGNED NOT NULL,
   data_time DATETIME(6) NOT NULL,
   recv_time DATETIME(6) NOT NULL,
   value_r TINYINT UNSIGNED DEFAULT NULL,
   quality TINYINT(1) DEFAULT NULL,
   att_error_desc_id INT UNSIGNED NULL DEFAULT NUL
 PRIMARY KEY(att_conf_id, data_time)
) ENGINE=InnoDB
PARTITION BY RANGE COLUMNS(data_time) (
      PARTITION p000 VALUES LESS THAN ('2015-01-01'),
      PARTITION p2015 VALUES LESS THAN ('2016-01-01'),
      PARTITION p2016 VALUES LESS THAN ('2017-01-01'),
      PARTITION p2017 VALUES LESS THAN ('2018-01-01'),
      PARTITION p2018 VALUES LESS THAN ('2019-01-01'),
      PARTITION p2019 VALUES LESS THAN ('2020-01-01'),
      PARTITION future      VALUES LESS THAN MAXVALUE
);
```

In InnoDB primary key is a clustered index:
the data is physically ordered on disk first by att_conf_id, second by data_time.
The primary key's leaf nodes are the data.

HDB++ @ Elettra

InnoDB schema

```
CREATE TABLE IF NOT EXISTS att_scalar_devuchar_ro
(
   att_conf_id INT UNSIGNED NOT NULL,
   data_time DATETIME(6) NOT NULL,
   recv_time DATETIME(6) NOT NULL,
   value_r TINYINT UNSIGNED DEFAULT NULL,
   quality TINYINT(1) DEFAULT NULL,
   att_error_desc_id INT UNSIGNED NULL DEFAULT
 PRIMARY KEY(att_conf_id, data_time)
) ENGINE=InnoDB
PARTITION BY RANGE COLUMNS(data_time) (
     PARTITION p000 VALUES LESS THAN ('2015-01-01'),
     PARTITION p2015 VALUES LESS THAN ('2016-01-01'),
     PARTITION p2016 VALUES LESS THAN ('2017-01-01'),
     PARTITION p2017 VALUES LESS THAN ('2018-01-01'),
     PARTITION p2018 VALUES LESS THAN ('2019-01-01'),
     PARTITION p2019 VALUES LESS THAN ('2020-01-01'),
     PARTITION future      VALUES LESS THAN MAXVALUE
);
```

Partitioning key must be part of every unique key in the table. Easy (and performing) **to delete old** data dropping entire partitions. SELECT performance improved thanks to partition pruning.

HDB++ @ Elettra

InnoDB schema

```
CREATE TABLE IF NOT EXISTS att_scalar_devuchar_ro
(
   att_conf_id INT UNSIGNED NOT NULL,
   data_time DATETIME(6) NOT NULL,
   recv_time DATETIME(6) NOT NULL,
   value_r TINYINT UNSIGNED DEFAULT NULL,
   quality TINYINT(1) DEFAULT NULL,
   att_error_desc_id INT UNSIGNED NULL DEFAULT NULL,
 PRIMARY KEY(att_conf_id, data_time)
) ENGINE=InnoDB
PARTITION BY RANGE COLUMNS(data_time) (
     PARTITION p000 VALUES LESS THAN ('2015-01-01'),
     PARTITION p2015 VALUES LESS THAN ('2016-01-01'),
     PARTITION p2016 VALUES LESS THAN ('2017-01-01'),
     PARTITION p2017 VALUES LESS THAN ('2018-01-01'),
     PARTITION p2018 VALUES LESS THAN ('2019-01-01'),
     PARTITION p2019 VALUES LESS THAN ('2020-01-01'),
     PARTITION future      VALUES LESS THAN MAXVALUE
);
```

Define partition also for unexpected values (data_time can be manually set in the Tango device!)

HDB++ @ Elettra

InnoDB partitions

Size of last partitions:

att_scalar_devdouble_rw#P#p2019.ibd	6,564,085,760 bytes
att_scalar_devlong_ro#P#p2019.ibd	6,035,603,456 bytes
att_scalar_devdouble_ro#P#p2019_09_10.ibd	5,062,524,928 bytes
att_scalar_devulong64_ro#P#p2019.ibd	3,539,992,576 bytes
att_scalar_devfloat_ro#P#p2019.ibd	3,061,841,920 bytes
att_scalar_devlong_rw#P#p2019.ibd	1,535,115,264 bytes
att_scalar_devstate_ro#P#p2019.ibd	1,329,594,368 bytes
att_scalar_devboolean_ro#P#p2019.ibd	796,917,760 bytes
att_array_devdouble_ro#P#p2019.ibd	675,282,944 bytes
att_scalar_devulong_ro#P#p2019.ibd	599,785,472 bytes
att_array_devlong_rw#P#p2019.ibd	490,733,568 bytes
att_array_devdouble_rw#P#p2019.ibd	213,909,504 bytes

…

----> total size of last partitions ~30 GB

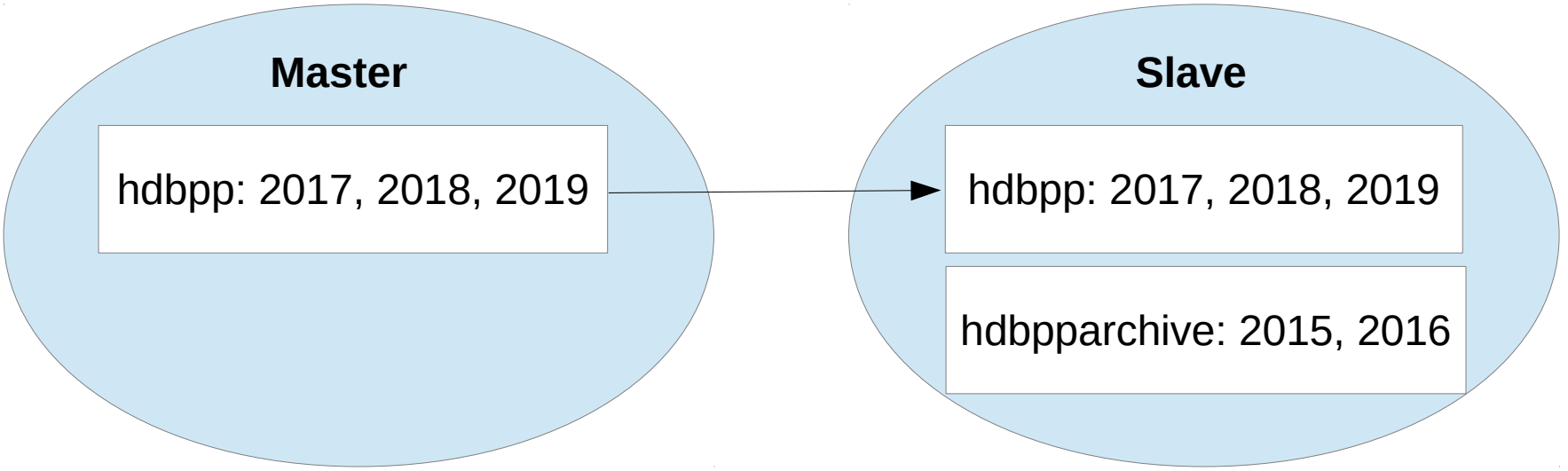
size of the whole DataBase **~350 GB**

Server RAM = 64 GB (50 GB InnoDB buffer pool) is enough to query really fast most recent data

HDB++ @ Elettra

InnoDB partitions

- Started archiving in 2015
- Keeping online current plus past 2 years
- Old partitions available read-only on the replica



- **hdbpp** size on disk: **350 GB**
- **hdbpparchive** size on disk: **146 GB**

HDB++ @ Elettra

InnoDB partitions

Manage partitions

• Need to create 2 tables for each type (one without partitioning)
CREATE TABLE hdbpparchive.att_scalar_devboolean_ro_tmp LIKE hdbpp.att_scalar_devboolean_ro;
ALTER TABLE hdbpparchive.att_scalar_devboolean_ro_tmp REMOVE PARTITIONING;
CREATE TABLE hdbpparchive.att_scalar_devboolean_ro LIKE hdbpp.att_scalar_devboolean_ro;

• Move partition in 2 steps
ALTER TABLE **hdbpp.att_scalar_devboolean_ro** EXCHANGE PARTITION p2017 WITH TABLE **hdbpparchive.att_scalar_devboolean_ro_tmp** WITHOUT VALIDATION;
ALTER TABLE **hdbpparchive.att_scalar_devboolean_ro** EXCHANGE PARTITION p2017 WITH TABLE **hdbpparchive.att_scalar_devboolean_ro_tmp** WITHOUT VALIDATION;

• Repeats moving for each partition

HDB++ @ Elettra

InnoDB partitions

Manage partitions

- Truncate partition moved

```
ALTER TABLE hdbpp.att_scalar_devboolean_ro TRUNCATE PARTITION p2017;
```

- Reorganize old partitions

```
ALTER TABLE hdbpp.att_scalar_devboolean_ro REORGANIZE PARTITION p000, p2017
  INTO
(PARTITION p000 VALUES LESS THAN ('2018-01-01'));
```

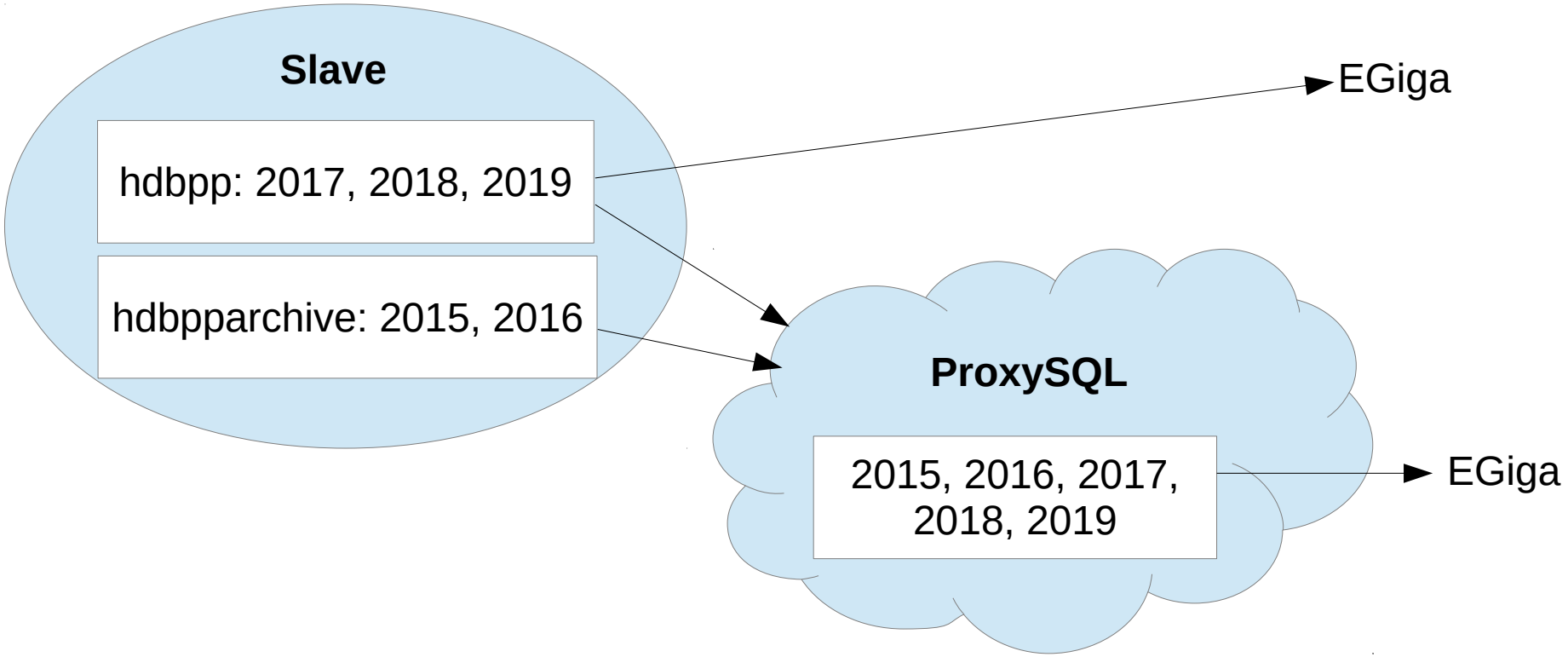
- Reorganize future partition

```
ALTER TABLE att_array_devboolean_ro REORGANIZE PARTITION future INTO
(
        PARTITION p2020 VALUES LESS THAN ('2021-01-01'),
        PARTITION future      VALUES LESS THAN MAXVALUE
);
```

HDB++ @ Elettra

Fermi

- By default Egiga reads hdbpp database on the slave
  (available data starting 2017-01-01)
- To query all data -> ProxySQL (https://proxysql.com)
- In the future hdbpparchive could be moved transparently to a different host



HDB++ @ Elettra

InnoDB

Partitioning issues and limits

- Extra work to manage partitions, done manually at the moment, to be automated in the future

- Some partitioning operation could take a write table lock. Safer to stop archiviation during maintenance to avoid running out of memory (hdbppes-srv buffering data arriving)

- Maximum number of partitions per table is 8192

- Foreign Key not supported on InnoDB partitioned tables

HDB++ @ Elettra

InnoDB

Improving INSERT performances

- Tune mysql configuration (by default configured for data safety and not performance):
  - innodb_flush_log_at_trx_commit != 1
    With the default setting of 1, logs are written and flushed to disk at each transaction. With 0, logs are written and flushed to disk once per second. With 2, logs are written at each transaction, but flushed to disk once per second. With 0 and 2 one second of transactions can be lost in case of crash.
  - sync_binlog != 1
    With the setting of 1 binary logs are syncronized to disk before every transaction. With 0 synchronization is disabled, flush to disk depends on OS. With N, the binary log is synchronized to disk every N writes. With settings != 1 increase the risk of data loss in case of crash or power failure.
  - innodb_doublewrite = 0
    Disable double buffering, enabled by default. If enabled data pages are written twice in order to be recovered in case of crash in the middle of a page write.
  - innodb_write_io_threads > 1
    Default value is 4, could be increased depending on HW.

HDB++ @ Elettra

InnoDB

Improving INSERT performances

- Since by default every INSERT is a transaction, collect multiple INSERT to be enclosed in a single transaction with BEGIN … COMMIT
  - Needs to be implemented in libhdbpp-mysql
  - Could also be implemented at hdbpp-es level so other transactional DB (PostgreSQL, TimescaleDB, ...) can take advantage of it.

HDB++ @ Elettra

Questions ?

HDB++ @ Elettra

ProxySQL

- Query caching: query results can be cached for a configurable timespan

- Application layer proxy: does not forward traffic blindly but it understand MySQL protocol and act accordingly

- Support Failover: handle connection to hundreds of backend servers

- Query routing: different classes of queries can be routed to different MySQL backend / clusters

- Query rewrite: using regex, matching queries can be rewritten

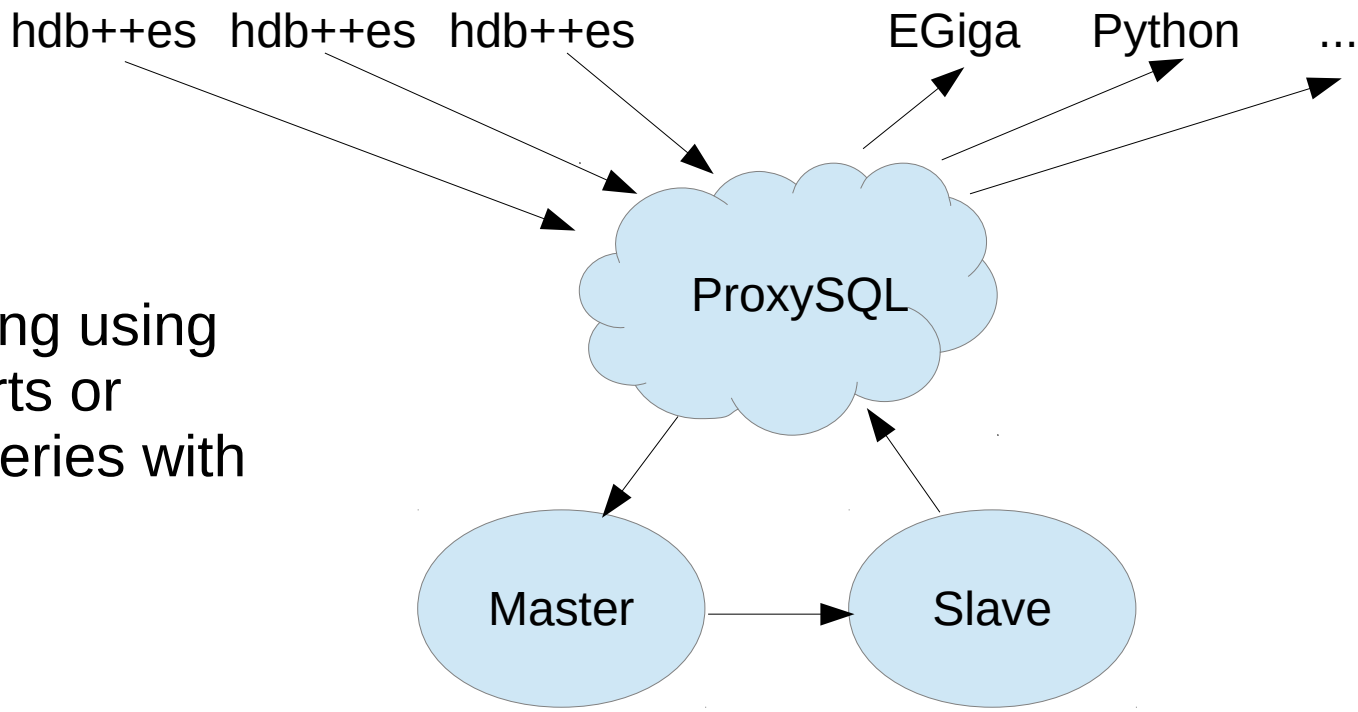
- Advanced topology support: proxies can be cascaded to obtain complex topologies

- ...

HDB++ @ Elettra

ProxySQL

- ProxySQL can make backend configuration and topology transparent to the application layer
- For example read/write split could be hidden to HDB++:



- R/W splitting using different ports or rewriting queries with regex

HDB++ @ Elettra

ProxySQL

- **Query rewrite:**
  To make EGiga query transparently data coming from hdbpp and hdbpparchive databases, regex rules matching all possible EGiga queries has been added to ProxySQL

Example: match year 2015 or 2016 and rewrite query reading from hdbpparchive instead of hdbpp

match_pattern:
'^SELECT data_time, value_r AS val FROM ([^.]*) WHERE att_conf_id=(\d+) AND data_time > "201([**56**])-(.*)" AND data_time < "201([**56**])-(.*)" ORDER BY (.*)$'

replace_pattern:
'SELECT data_time, value_r AS val FROM **hdbpparchive**.\1 WHERE att_conf_id=\2 AND data_time > "201\3-\4" AND data_time < "201\5-\6" ORDER BY \7'