УНИВЕРСИТЕТ ИТМО

ClickHouse: MPP database in Big Data world

Nikolay Butakov, Azamat Gainetdinov, Sergey Teryoshkin

What is ClickHouse ?

ITMO UNIVERSITY

- Distributed column-oriented MPP database supporting SQL dialect and written in C++

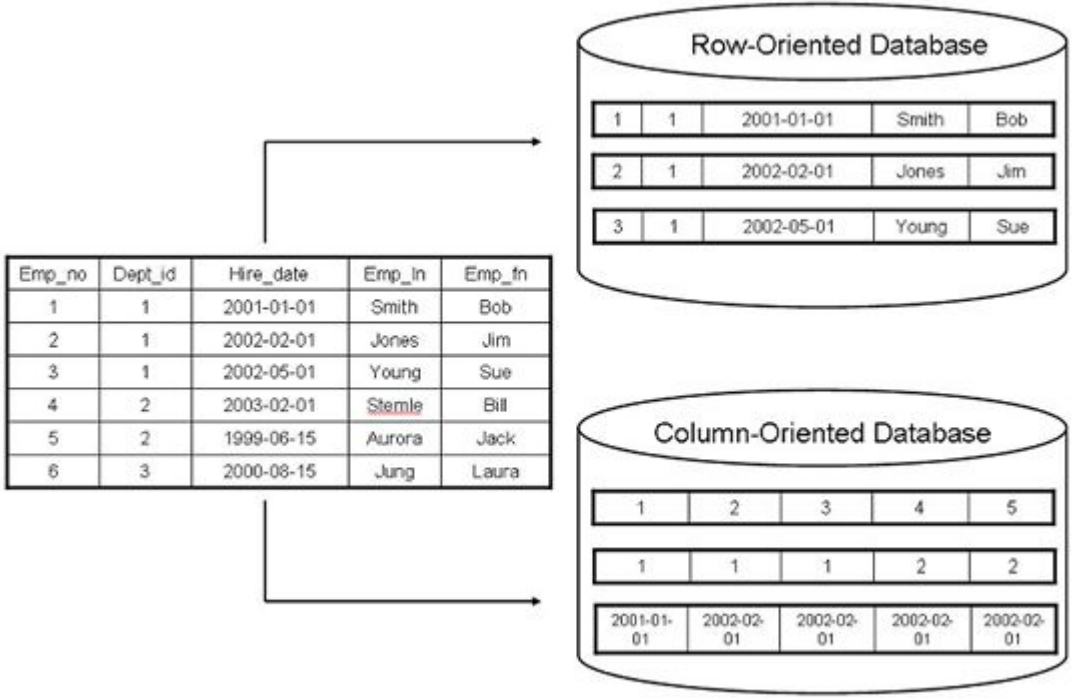
What is it for ?

- OLAP
- Monitoring and events
- TimeSeries
- …. any case where you need to calculate aggregations on huge volume of data

ITMO more than a UNIVERSITY

Column-oriented db

- Physical layout may be *row-wise* or *column-wise*
- Layout defines *spatial* on-disk location and data proximity
- Layout affects effectivity of certain data-access patterns and heavily affects compression



Emp_no	Dept_id	Hire_date	Emp_ln	Emp_fn
1	1	2001-01-01	Smith	Bob
2	1	2002-02-01	Jones	Jim
3	1	2002-05-01	Young	Sue
4	2	2003-02-01	Stemle	Bill
5	2	1999-06-15	Aurora	Jack
6	3	2000-08-15	Jung	Laura

https://www.dbbest.com/blog/column-oriented-database-technologies/

Row-oriented vs Column-oriented

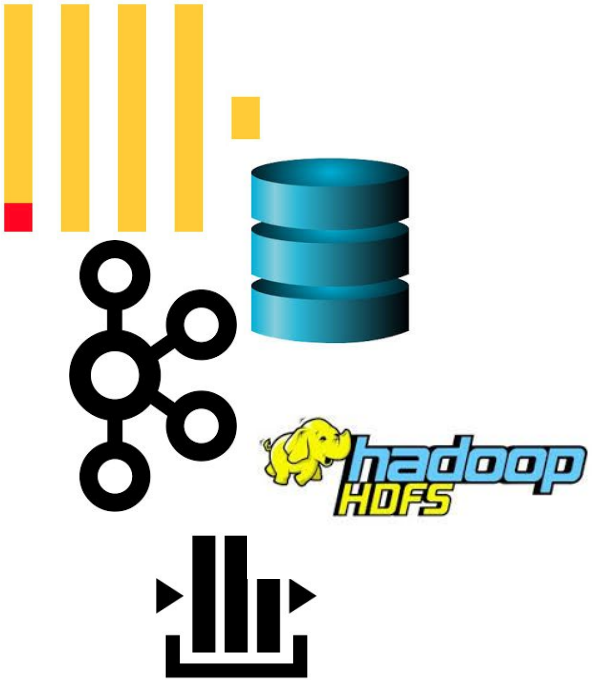
ITMO UNIVERSITY

Row oriented data stores	Column oriented data stores
Data is stored and retrieved one row at a time and hence could read unnecessary data if some of the data in a row are required.	In this type of data stores, data are stored and retrieve in columns and hence it can only able to read only the relevant data if required.
Records in Row Oriented Data stores are easy to read and write.	In this type of data stores, read and write operations are slower as compared to row-oriented.
Row-oriented data stores are best suited for online transaction system.	Column-oriented stores are best suited for online analytical processing.
These are not efficient in performing operations applicable to the entire datasets and hence aggregation in row-oriented is an expensive job or operations.	These are efficient in performing operations applicable to the entire dataset and hence enables aggregation over many rows and columns.
Typical compression mechanisms which provide less efficient result than what we achieve from column-oriented data stores.	These type of data stores basically permits high compression rates due to little distinct or unique values in columns.

Table engines

ITMO UNIVERSITY

- ClickHouse has multiple table engines: MergeTree, Buffer, File, Kafka, HDFS, etc.
- Engine has direct implications on performance of your queries and logical gurantes
- Different tables inside same database may have different engines
- It is a common practice to combine engines
- Main engine is MergeTree and its derivatives. It provides columnar storage, index ability and other features.



ITMOre than a UNIVERSITY

MergeTree engine

ITMO UNIVERSITY

What is MergeTree ?

- special kind of data structure to store data on-disk, *manage* their lifecycle and *optimize* them for reading through time

Why do we need MergeTree ?

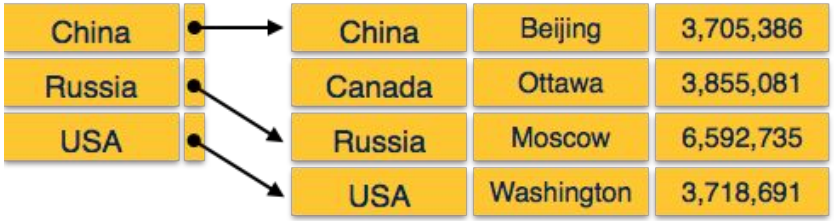
- writes and reads performance conflicts as they require different data structures to work better

ClickHouse MergeTree makes it optimized to benefit huge analytical aggregations, e.g.:

- long sequential scans
- huge batches of writes

ITMO more than a UNIVERSITY

What is sparse index?



Sparse Index:

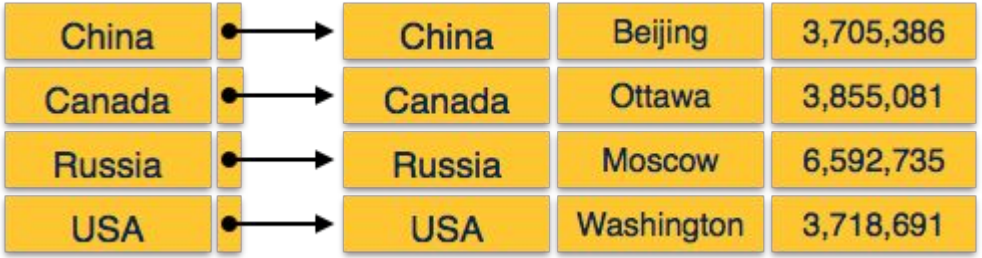
- Index entries only for ranges, not for records.
- To locate a record, we find the index record with the largest search key value less than or equal to the search key value we are looking for.
- We read all records starting with this found record and until we find the desired record.

https://www.tutorialspoint.com/dbms/dbms_indexing.htm

Dense index vs Sparse index

ITMO UNIVERSITY

Dense index

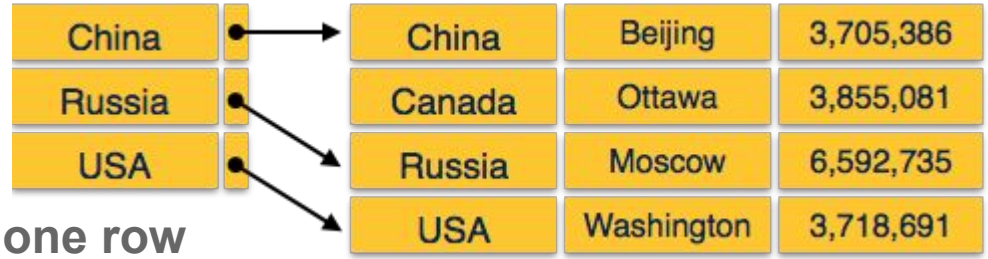


Dense index record appears for every search key value in file.

What to do when you have 10 billions of records and more is coming?

Sparse index record appears for a range of values and that can fit the RAM in most cases.

Sparse index



No uniqueness for primary key, not possible to address exactly one row

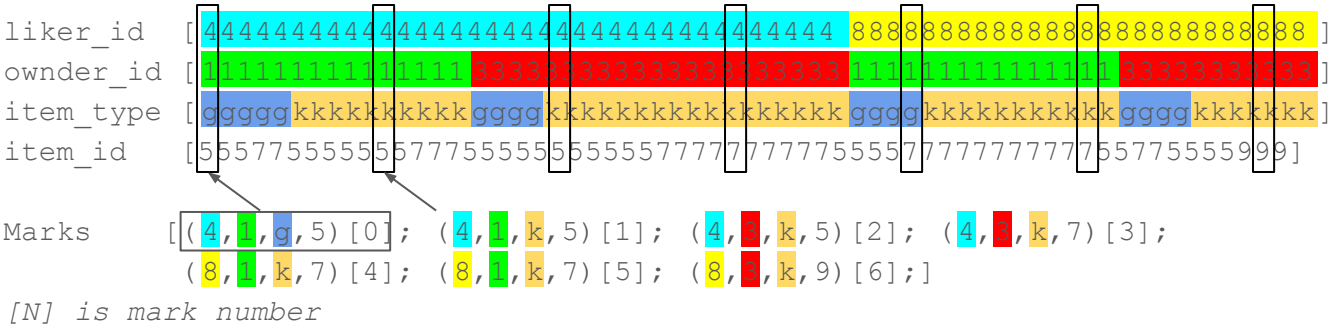
ITMO more than a UNIVERSITY

=> **dense indices** are *faster* in general, but **sparse indices** *require less space* and impose *less maintenance* for insertions and deletions

https://www.tutorialspoint.com/dbms/dbms_indexing.htm

Primary Key and Sorting

Likes table has (liker_id, owner_id, item_type, item_id) as a primary key. In this case, the sorting and index can be illustrated as follows:



If the data query specifies:

- liker_id = 4, the server reads the data in the range of marks [0, 4).
- liker_id = 4 AND owner_id = 3, the server reads the data in the range of marks [1, 4).
- Item_type = ‘g’, the server reads data in range of marks [0, 2) U [3, 4) U [5, <end>].

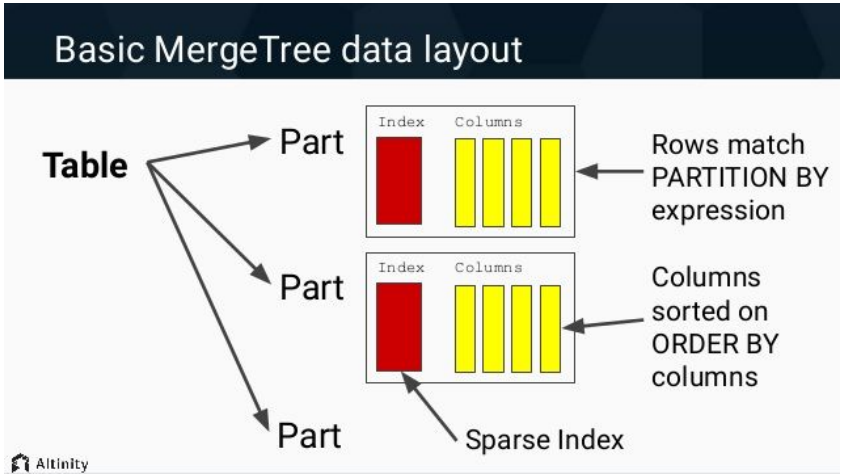
The examples above show that it is always more effective to use an index than a full scan.

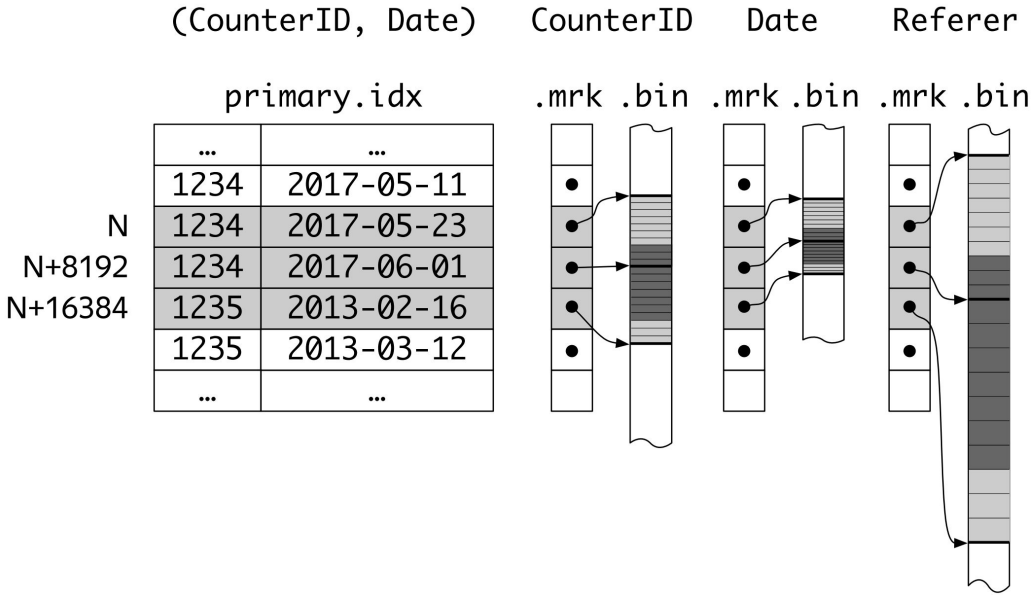
Physical data layout

ITMO UNIVERSITY

For each part, ClickHouse stores:

- files of columns .bin (each column in separate file)
- .mrk file for each column
- .idx file of primary index

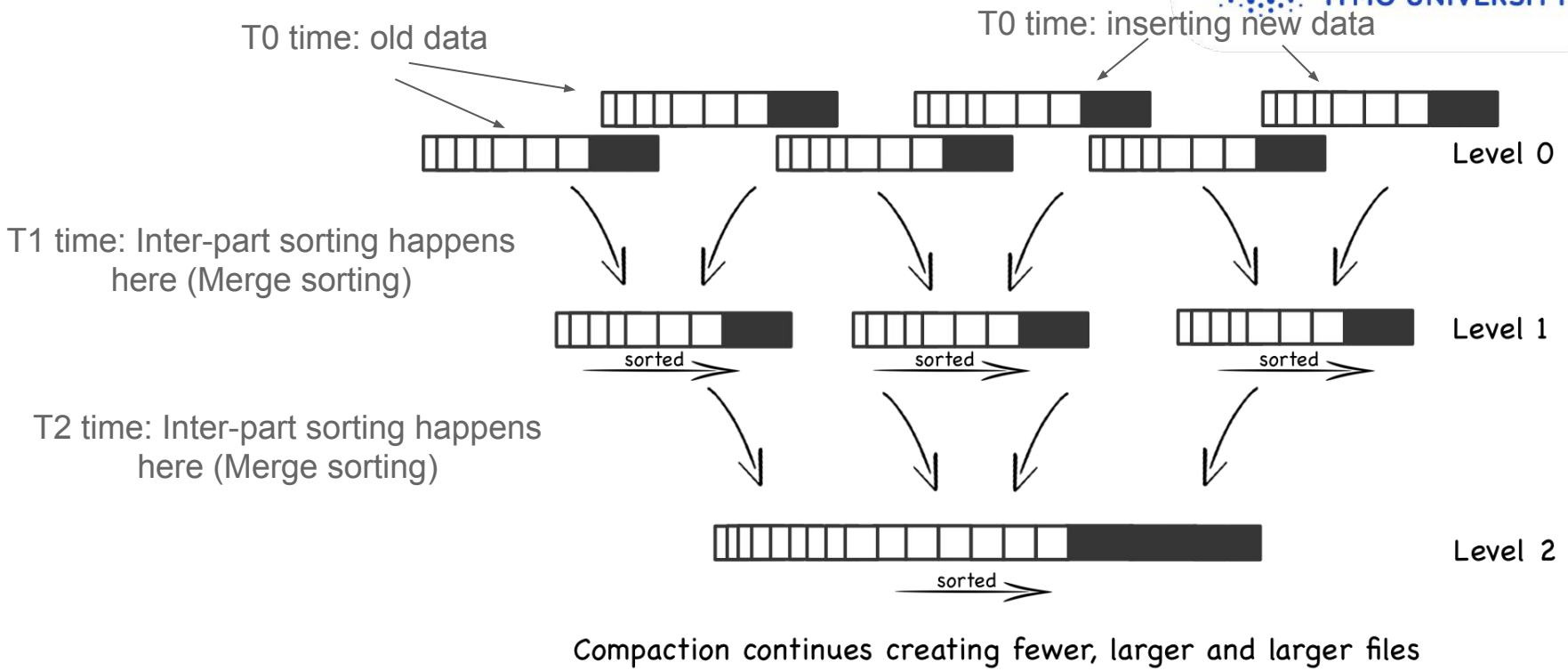




ITMO more than a UNIVERSITY

https://www.altinity.com/
https://www.slideshare.net/profyclub_ru/clickhouse-77412233

MergeTree merge processes



https://en.wikipedia.org/wiki/Log-structured_merge-tree

MergeTree features

ITMO UNIVERSITY

Advantages:

- Keeps data sorted to make efficient on-disk data locating (using offsets) and sequential data reading (performance may degraded slightly after writing but improves over time)
- Allows to write data quickly without expensive data reallocating (no copying operations on write)

Disadvantages:

- No updates, each writing goes to separate file.
- ClickHouse does not like writing of separate values or many small batches because many small files degraded filesystem performance and increase randomness of read.

ITMO more than a UNIVERSITY

MergeTree engine family

ITMO UNIVERSITY

Modifications of MergeTree can partially handle its limitations exploiting merge phase:

- ReplacingMergeTree - remove duplicated rows by sorting key. Remains row with largest value of version-column.
- CollapsingMergeTree - remove duplicated rows preserving no more than two versions different by state
- AggregatingMergeTree - aggregates duplicated rows by applying aggregating functions written with corresponding GROUP BY clause
- SummingMergeTree - aggregates duplicated rows by simply summing their non-key fields
- VersionedCollapsingMergeTree - CollapsingMergeTree with multiple versions.

ITMO more than a UNIVERSITY

Create table syntax

ITMO UNIVERSITY

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER]
(
     name1 ColumnType [DEFAULT|MATERIALIZED|ALIAS expr1] [CODEC] [TTL ttl_expr],
     name2 ColumnType [DEFAULT|MATERIALIZED|ALIAS expr1] [CODEC] [TTL ttl_expr],
     ....
)
ENGINE = EngineName([options])
[PARTITION BY expr]
[ORDER BY expr]
[SAMPLE BY expr]
[TTL expr]
[SETTINGS property1 = value,
          property2 = value,
          … ]
```

ITMO more than a UNIVERSITY

Table example: user_profiles

```
CREATE TABLE vk.user_profiles
(
    `about` Nullable(String),
    `activities` Nullable(String),
    `bdate` Nullable(String),
    `blacklisted` Int8,
    `books` Nullable(String),
    `can_post` Int8,
    `can_see_all_posts` Int8,
    `can_see_audio` Int8,
    `can_send_friend_request` Int8,
    `can_write_private_message` Int8,
    `career` Nested(
    city_id Nullable(UInt32),
    company Nullable(String),
    country_id Nullable(UInt32),
    from Nullable(UInt32),
    group_id Nullable(UInt32),
    position Nullable(String),
    until Nullable(UInt32)),
    `city_id` Int32,
     ...
    `nickname` Nullable(String),
    `occupation_id` Int64,
    `occupation_name` Nullable(String),
    `occupation_type` Int8,
    `online` Int8,
    `online_app` Int32,
    `online_mobile` Int8,
    `personal_alcohol` Int8,
    `personal_inspired_by` Nullable(String),
    `personal_langs` Array(Nullable(String)),
    `personal_life_main` Int8,
    `personal_people_main` Int8,
    `personal_political` Int16,
    `personal_religion` Nullable(String),
     ...
    `university` Int32,
    `university_name` Nullable(String),
    `verified` Int8,
    `wall_comments` Int8
)
ENGINE = ReplacingMergeTree(ctime)
PARTITION BY (sex, deactivated)
ORDER BY id
```

Total columns (include Nested fields): 149

```
SELECT id, first_name, last_name,   career.company, career.from,
    career.until, career.position FROM vk.user_profiles
LEFT ARRAY JOIN career
WHERE (deactivated < 0) AND isNotNull(career.company) AND
isNotNull(career.from) AND isNotNull(career.position)
```

id	first_name	last_name	career.company	career.from	career.until	career.position
6223293	Lyokha	Slyonik	рампы	2006	2007	скейтер-бэймер
6227057	Sergey	Malkov	ВЧ 2375 командир ПСКР 656	1977	1991	Командир пограничного корабля
6227245	Ledzhar	Khavochik	Канализация	1980	2008	насяльника
6228969	Denis	Mescheryakov	Universität Mainz	2007	NULL	Wissenschaftlicher Mitarbeiter
6229025	Pavel	Bashkirtsev	ОАО "УралЭлектроМонтаж"	2008	NULL	Электромонтажник по силовым сетям
6229883	Yury	Nikonchuk	IMC	2006	NULL	Integration Engineer
6230231	Alexander	Azarov	Корпорация ЗЛА	2015	NULL	Спасение мира от доброты
6230557	Dima	Rupasov	Дворец Султана	1984	2009	принц брунейский
6231343	Alexander	Molchanov	Связьконструкция	2001	2005	инженер-связи
6232661	Nick	Pilipenko	ННЦ ХФТИ	2007	NULL	мл. научный сотрудник
6234057	Utko	Lokh	Озерки,первое озеро	2007	2009	жопу подставляю=(((а чё ??? деньги и мне нужны
6234117	Gey	Zhoporvatel	больница	1979	NULL	ПРОКТОЛОГ-ЖОПОРВАТЕЛЬ
6236205	Anatoly	Marchuk	inetcom	2005	2009	сасадмин
6237905	Alexander	Zhevanov	ArtNet	2007	NULL	директор
6237963	Mikhail	Suomolaynen	ОТЗ	1978	1978	строгальщик
6237963	Mikhail	Suomolaynen	завод "Светлана"	1978	1979	радиомонтажник
6237963	Mikhail	Suomolaynen	институт биологии КФАН	1984	1994	каких только не было
6238525	Alexander	Lazun	Hitech	2004	NULL	Senior system & security administrator
6239389	Sergey	Gorodetsky	ЗАО "Холодон"	2008	2008	Монтажник
6239389	Sergey	Gorodetsky	ОАО "РАТОН"	2008	2011	Инженер-конструктор
6239389	Sergey	Gorodetsky	ООО "Афелд Групп"	2011	2013	Зам. директора
6239519	Kostya	Osipov	ЗАО "Тандер" (Сеть магазинов "Магнит")	2007	2008	Старший Системотехник, Начальник отдела IT
6239519	Kostya	Osipov	OPCK.INFO	2008	NULL	Ведущий проекта
6239519	Kostya	Osipov	ОАО "Уфанет"	2008	2012	Системный администратор
6239519	Kostya	Osipov	fotostrana.ru	2012	2013	PHP программист
6239519	Kostya	Osipov	Ситилинк	2013	NULL	PHP-программист
6240429	Ilya	Guk	ОАО ОЭМК	1997	2008	мастер по ремонту оборудования
6240509	Maxim	Bagino	Стройка	1991	2009	Чернораб
6240509	Maxim	Bagino	Минирынок Уручье-3	2007	2009	Продавец-консультант
6240509	Maxim	Bagino	ЗАО "ЗапСиб"- Монтажников 39)	2010	2011	Кладовщик
6240509	Maxim	Bagino	Магазин "5 Элемент"	2011	2014	Продавец консультант
6240551	Alexey	Shtefan	uuuuuuuuu	1955	1956	))))))))))
6242033	Roman	Stepanov	ОАО "НПК"СПП"	2012	NULL	Начальник сектора
6242125	Alexander	Ovcharenko	Шоколадная компания "Barens Chocolate"	2007	2010	Инженер-технолог
6242173	Maxim	Perepilitsa	Процессинговый центр agebill.com	2007	NULL	Ген. Директор
6242197	Denis	Utkin	РосНИПЧИ Микроб	1996	NULL	ведущий научный сотрудник
6243749	Igor	Bosnyakov	ЦАГИ	2008	NULL	нач.секты

Column encodings and compression

ITMO UNIVERSITY

ClickHouse supports following compression algorithms: LZ4 (used by default), LZ4HC and ZSTD.
Columns may be encoded before compression there are many encodings:

- Delta
- DoubleDelta
- Gorilla
- T64

Encodings and compression can be specified per column.

The example below defines DoubleDelta encoding for a column. Note, that it turns off default compression.

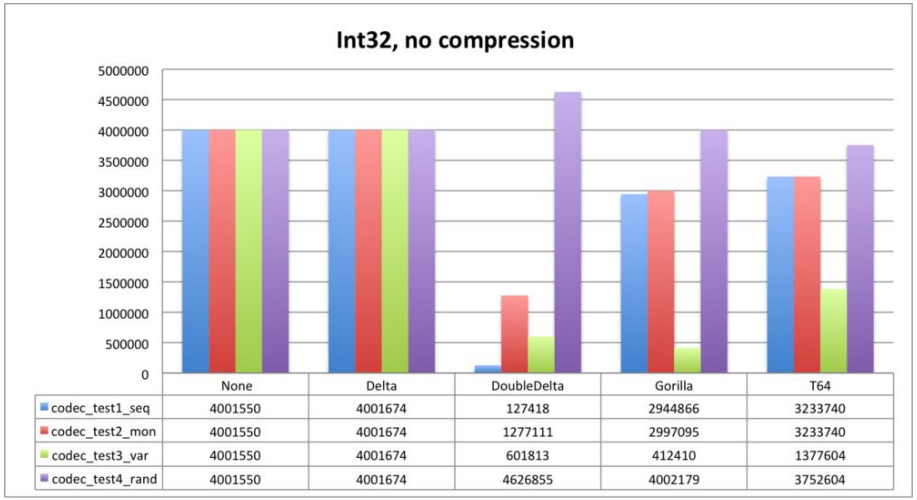
```
ts DateTime Codec(DoubleDelta) -- encoded but NOT compressed
```

In order to have both encoding and compression, codecs can be chained, as shown below:

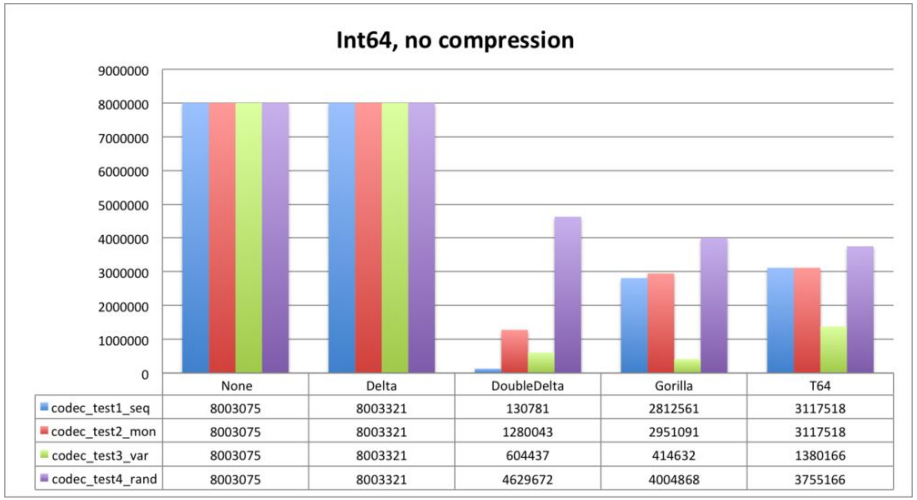
```
ts DateTime Codec(DoubleDelta, ZSTD) -- encoded AND compressed
```

ITMO More than a UNIVERSITY

Encoding and Compression Benchmarks



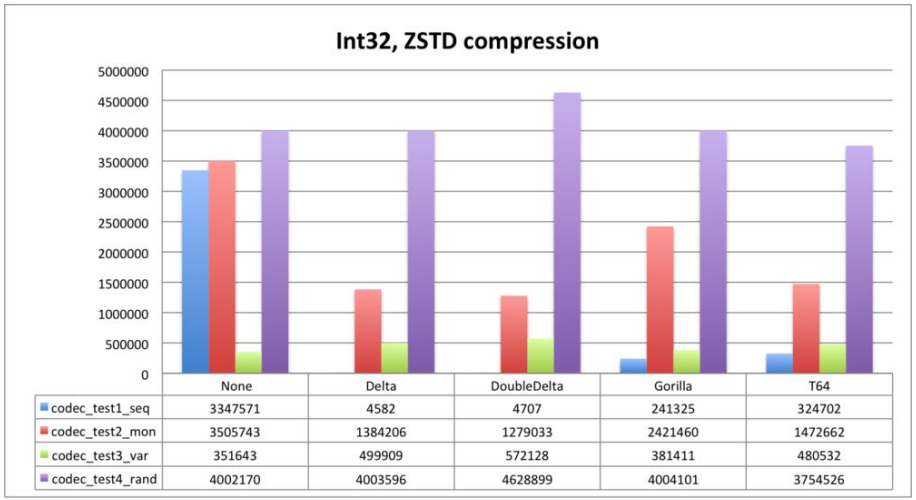
	None	Delta	DoubleDelta	Gorilla	T64
codec_test1_seq	4001550	4001674	127418	2944866	3233740
codec_test2_mon	4001550	4001674	1277111	2997095	3233740
codec_test3_var	4001550	4001674	601813	412410	1377604
codec_test4_rand	4001550	4001674	4626855	4002179	3752604



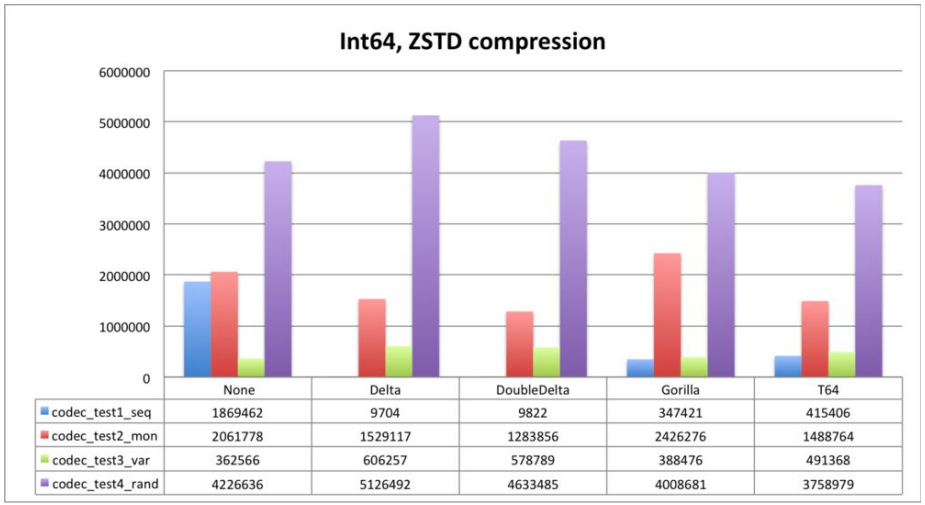
	None	Delta	DoubleDelta	Gorilla	T64
codec_test1_seq	8003075	8003321	130781	2812561	3117518
codec_test2_mon	8003075	8003321	1280043	2951091	3117518
codec_test3_var	8003075	8003321	604437	414632	1380166
codec_test4_rand	8003075	8003321	4629672	4004868	3755166

https://www.altinity.com/blog/2019/7/new-encodings-to-improve-clickhouse

Encoding and Compression Benchmarks



	None	Delta	DoubleDelta	Gorilla	T64
codec_test1_seq	3347571	4582	4707	241325	324702
codec_test2_mon	3505743	1384206	1279033	2421460	1472662
codec_test3_var	351643	499909	572128	381411	480532
codec_test4_rand	4002170	4003596	4628899	4004101	3754526

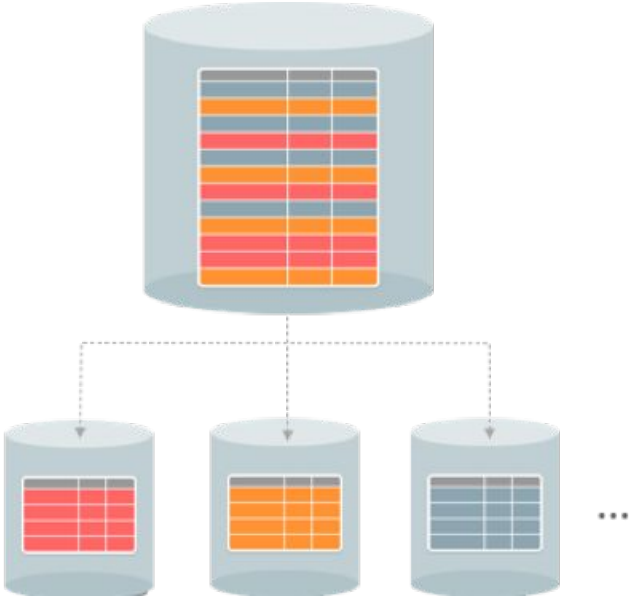


	None	Delta	DoubleDelta	Gorilla	T64
codec_test1_seq	1869462	9704	9822	347421	415406
codec_test2_mon	2061778	1529117	1283856	2426276	1488764
codec_test3_var	362566	606257	578789	388476	491368
codec_test4_rand	4226636	5126492	4633485	4008681	3758979

https://www.altinity.com/blog/2019/7/new-encodings-to-improve-clickhouse

Sharding and replication

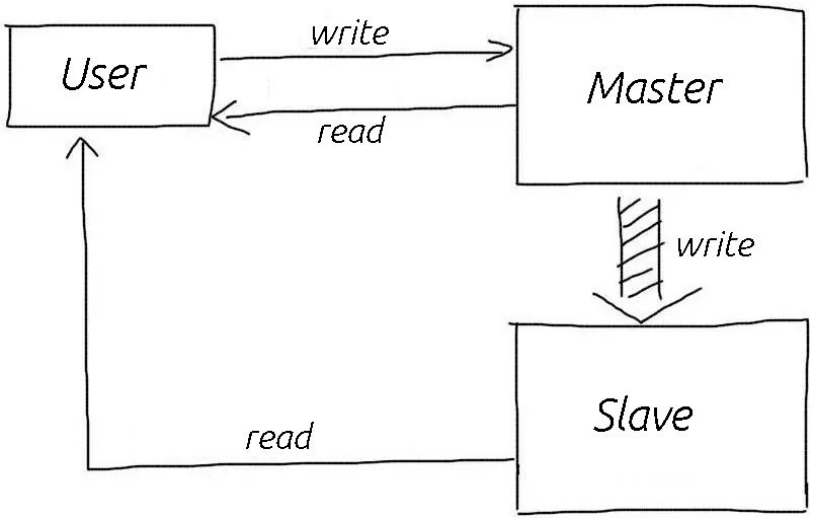
Sharding is especially useful when vertical scaling (increasing server capacity) is unprofitable or impossible.

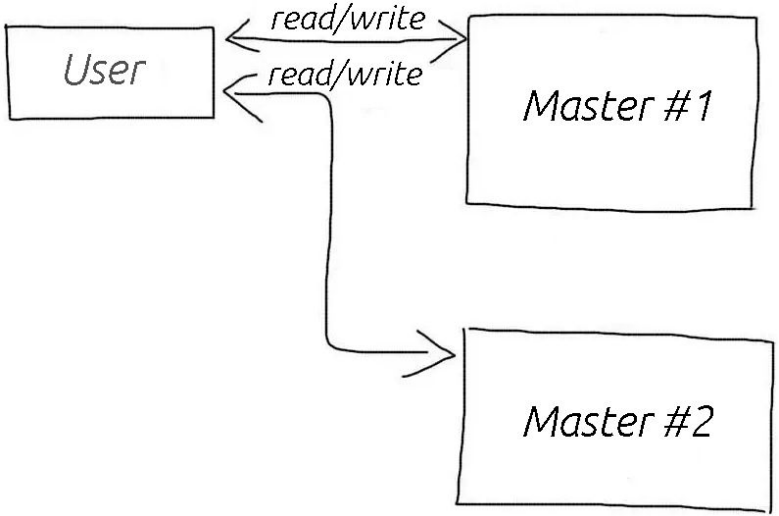


Sharding expression is deterministic function and using to distribute data across servers.
Sharding expression gets one or several fields of table's record and returns number of server for this record.
Sharding expression may be presented as hash function, simple function (like a % n), even as constant (all data will be loaded into one shard).
ClickHouse has significant amount of hash functions: sha*, javaHash, intHash*, sipHash*, MD5, murmur_Hash*, etc.

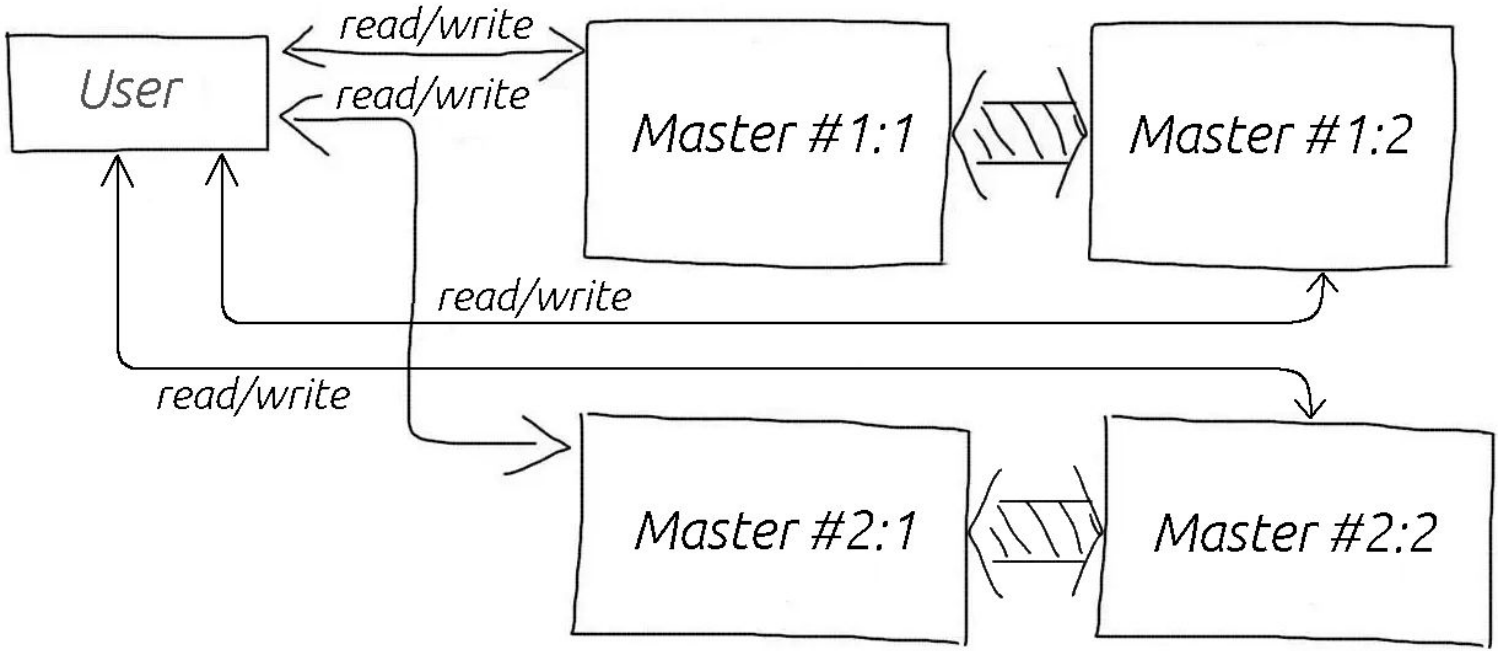
https://nageshbattula.com/2018/08/06/oracle-sharding-18c-blog-post-series-2-pdb-sharding/

Master-Master vs Master-Slave





Data replication



Why master-master?

ITMO UNIVERSITY

- User has more control on what he does thus it is possible to optimize some operations
- Master-master brings AP to ClickHouse from CAP-theorem.
  It means that you own system remains operable (can write and read) even if cluster become fractured. No single point of failure
- It is faster for writes (each node can process write requests, not only single master)

Disadvantages:

- Consistency is not an option (it is a pain).
  Writing / reading client should be responsible and aware for that.

ITMO more than a UNIVERSITY

Distributed storage and processing

ITMO UNIVERSITY

To handle queries that require data aggregating or retrieval from several nodes, ClickHouse uses special kind of table engine **Distributed**

Distributed engine:

- doesn’t store any data by itself
- it distributes inserts according predefined hashing function between tables on different nodes
- it queries tables on different nodes and merge results on a single node before sending results to client
- it may form queries with partial aggregations on individual nodes and fully aggregate all data on a single before sending results to client

ITMO more than a UNIVERSITY

Since ClickHouse have MASTER-MASTER architecture,
tables should be created into all servers of cluster

Distributed table: example

```
CREATE TABLE db.distr_table AS db.table
ENGINE = Distributed(some_cluster, db, table,
                     sharding_expr());
```

```
CREATE TABLE vk.distr_likes AS vk.likes
ENGINE = Distributed(test_cluster_two_shards, vk, likes, likerId % 2)
```

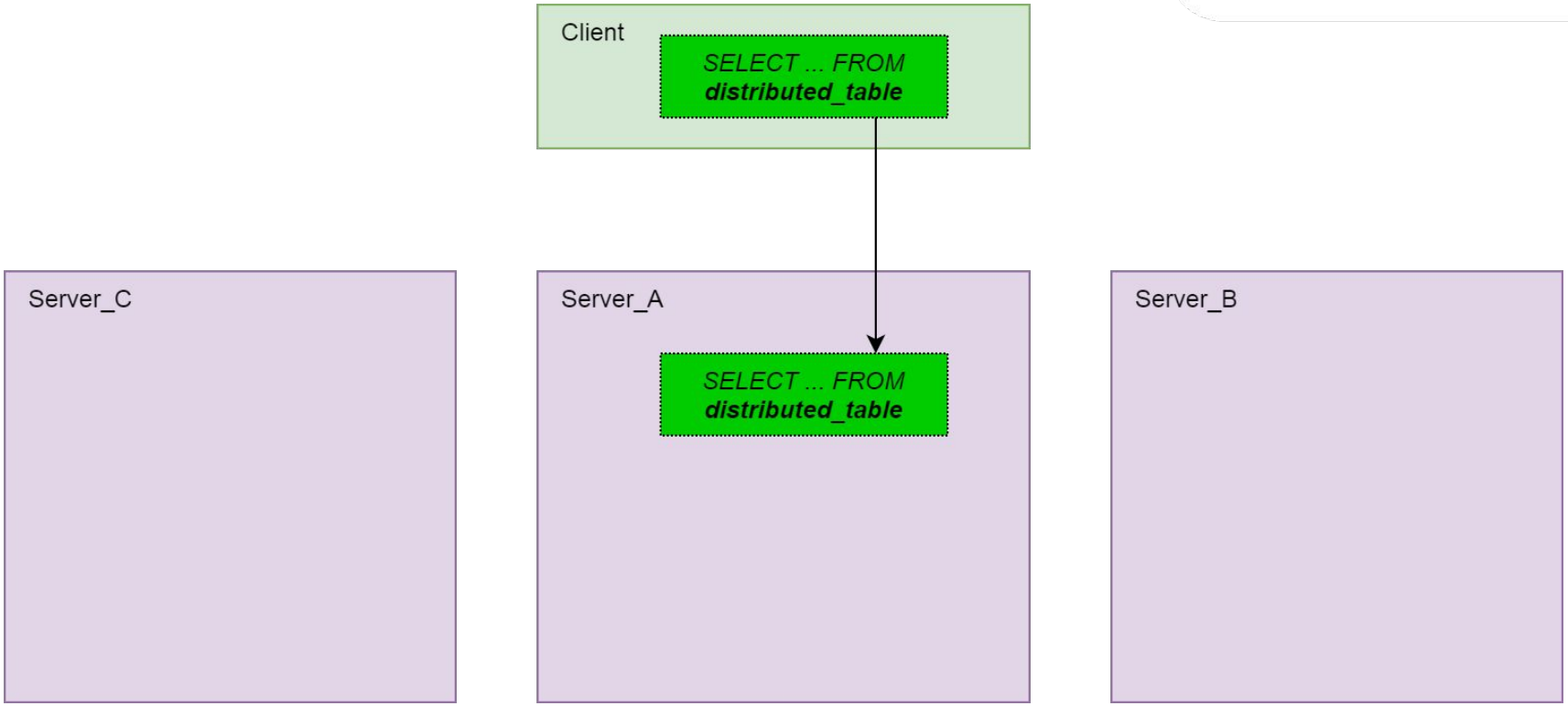
```
(base) sergey@sergey-pc:~$ clickhouse-client --multiline
ClickHouse client version 19.15.2.2 (official build).
Connecting to localhost:9000 as user default.
Connected to ClickHouse server version 19.15.2 revision 54426.

sergey-pc :) select count() from vk.likes;

SELECT count()
FROM vk.likes

┌─count()─┐
│ 14333669 │
└─────────┘

1 rows in set. Elapsed: 0.042 sec. Processed 14.33 million rows, 57.33 MB (343.44 million rows/s., 1.37 GB/s.)

sergey-pc :) select count() from vk.distr_likes;

SELECT count()
FROM vk.distr_likes

┌─count()─┐
│ 28644579 │
└─────────┘

1 rows in set. Elapsed: 0.043 sec. Processed 28.64 m

sergey-pc :)
```

```
(base) sergey@sergey-pc:~$ clickhouse-client --multiline --host 10.253.1.99
ClickHouse client version 19.15.2.2 (official build).
Connecting to 10.253.1.99:9000 as user default.
Connected to ClickHouse server version 19.15.2 revision 54426.

ch-VirtualBox :) select count() from vk.likes;

SELECT count()
FROM vk.likes

┌─count()─┐
│ 14310910 │
└─────────┘

1 rows in set. Elapsed: 0.022 sec. Processed 14.31 million rows, 57.24 MB (662.13 million rows/s., 2.65 GB/s.)

ch-VirtualBox :) select count() from vk.distr_likes;

SELECT count()
FROM vk.distr_likes

┌─count()─┐
│ 28644579 │
└─────────┘

1 rows in set. Elapsed: 0.022 sec. Processed 28.64 million rows, 114.58 MB (1.28 billion rows/s., 5.10 GB/s.)

ch-VirtualBox :)
```

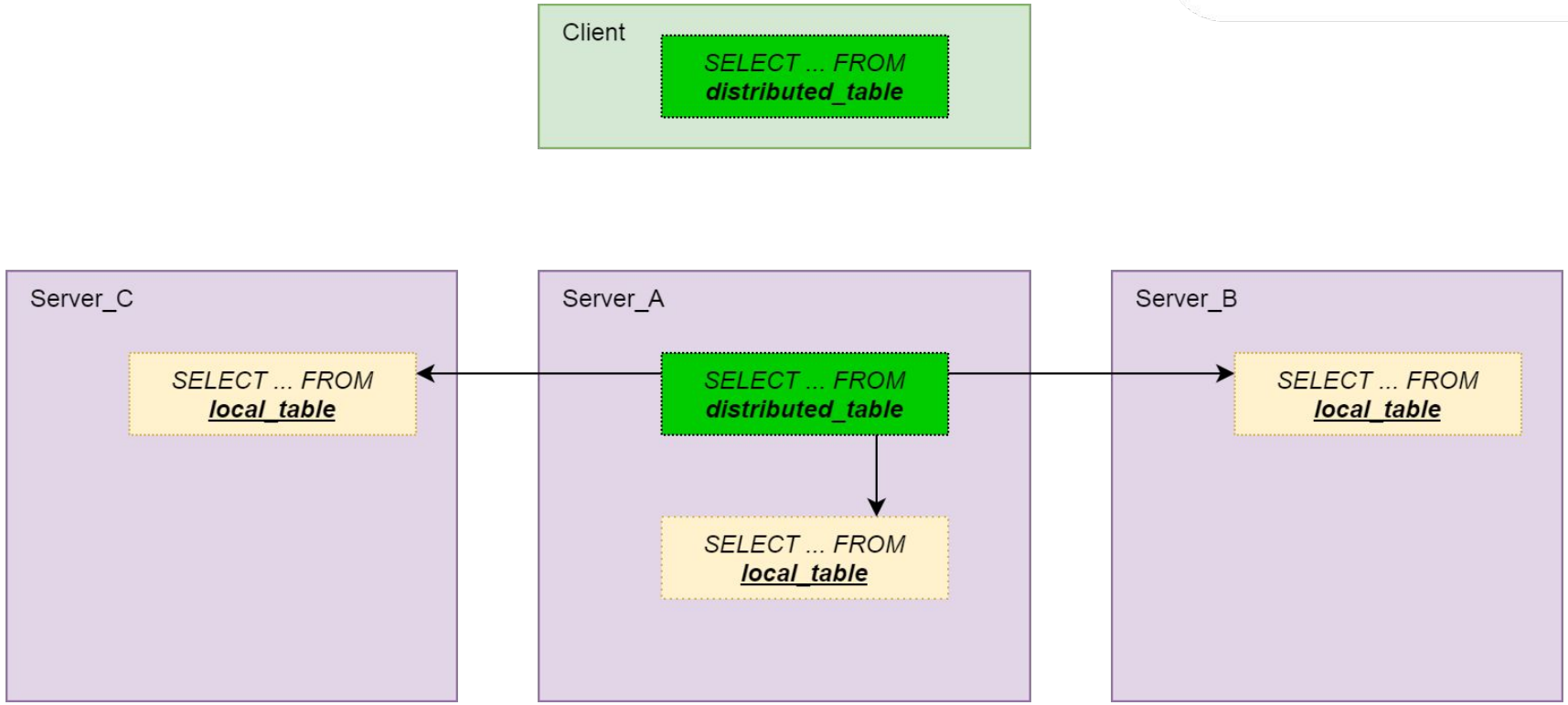
/etc/clickhouse-server/config.xml

```
<remote_servers>
    <test_cluster_two_shards>
        <shard>
            <weight>1</weight>
            <internal_replication>false</internal_replication>
            <replica>
                <host>10.253.1.99</host>
                <port>9000</port>
            </replica>
            <replica>
                <host>10.253.1.103</host>
                <port>9000</port>
            </replica>
        </shard>
        <shard>
            <weight>2</weight>
            <internal_replication>false</internal_replication>
            <replica>
                <host>10.253.1.53</host>
                <port>9000</port>
            </replica>
        </shard>
    </test_cluster_two_shards>
</remote_servers>
```

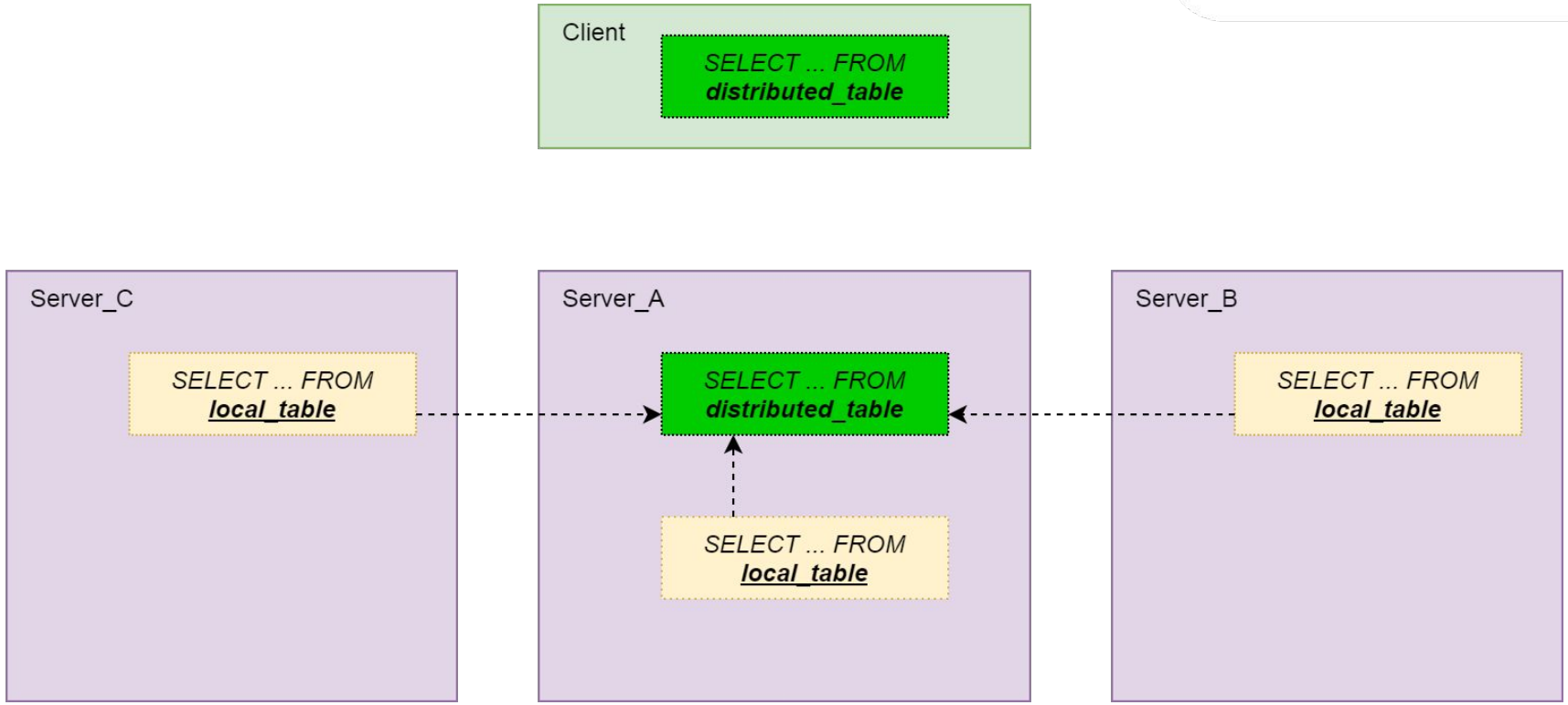
Distributed table: how it works?



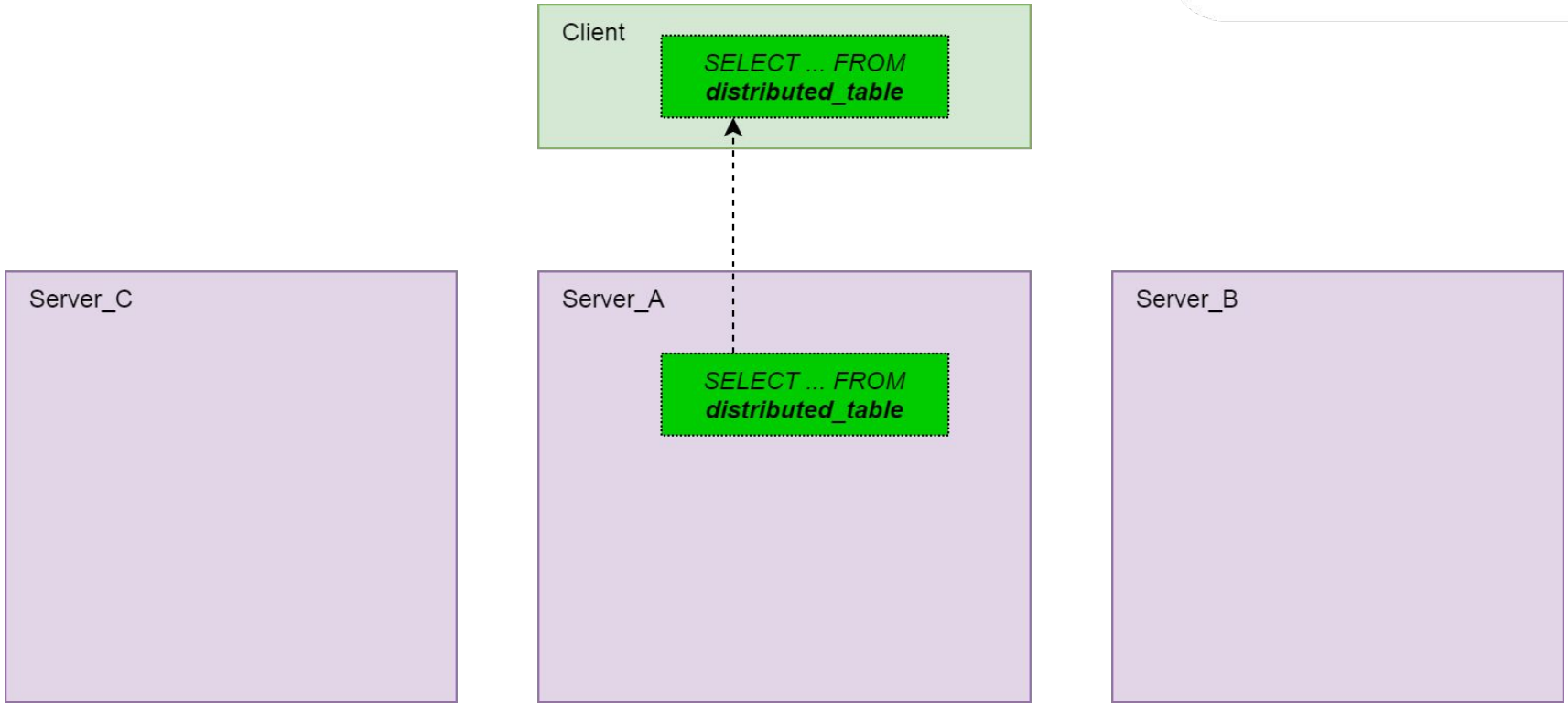
Distributed table: how it works?



Distributed table: how it works?



Distributed table: how it works?



Distributed table: data inserting

ITMO UNIVERSITY

There are two ways to insert data into Distributed tables:

- Insert data into local tables on shards (recommended). It allows to implement sharding based on complex logic, suitable for your application field.
- Directly insert into Distributed table with sharding expression. The simplest way, but it may lead to data skew.

ITMO more than a UNIVERSITY

Joins

ITMO UNIVERSITY

ClickHouse supports these types of joins:

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN
- CROSS JOIN

Available strictness rules:

- ALL
- ANY
- ASOF

Example:

```
SELECT l.user_id, p.post_id, p.text
FROM posts p
ANY LEFT JOIN likes l
ON (l.post_id = p.post_id)
```

ITMO more than a UNIVERSITY

Strictness rule: ANY

ITMO UNIVERSITY

id	val1	val2	val3
1	Ann	22	117
2	Felix	27	NULL
3	John	38	131

id	flag	timestamp
1	0	1587997365
1	1	1587999999
2	1	1587997000

ANY LEFT JOIN

id	val1	val2	val3	flag	timestamp
1	Ann	22	117	0	1587997365
2	Felix	27	NULL	1	1587997000
3	John	38	131	NULL	NULL

ITMO more than a UNIVERSITY

Strictness rule: ALL

id	val1	val2	val3
1	Ann	22	117
2	Felix	27	NULL
3	John	38	131

id	flag	timestamp
1	0	1587997365
1	1	1587999999
2	1	1587997000

ALL LEFT JOIN

id	val1	val2	val3	flag	timestamp
1	Ann	22	117	0	1587997365
1	Ann	22	117	1	1587999999
2	Felix	27	NULL	1	1587997000
3	John	38	131	NULL	NULL

Strictness rule: ASOF

user_id	like_id	like_datetime
17624	114	2020-04-18 16:53:27
17624	115	2020-04-18 18:27:07
316771	1314	2020-03-27 13:42:47

user_id	latitude	longitude	location_datetime
17624	59.957236	30.308415	2020-04-18 16:47:21
17624	59.955638	30.308553	2020-04-18 16:52:34
17624	60.038642	30.297955	2020-04-18 18:20:03
316771	55.746322	37.648310	2020-03-27 13:42:46
316771	55.745705	37.647296	2020-03-27 13:43:15

ASOF LEFT JOIN

user_id	like_id	latitude	longitude	location_datetime	like_datetime
17624	114	59.955638	30.308553	2020-04-18 16:52:34	2020-04-18 16:53:27
17624	115	60.038642	30.297955	2020-04-18 18:20:03	2020-04-18 18:27:07
316771	1314	55.746322	37.648310	2020-03-27 13:42:46	2020-03-27 13:42:47

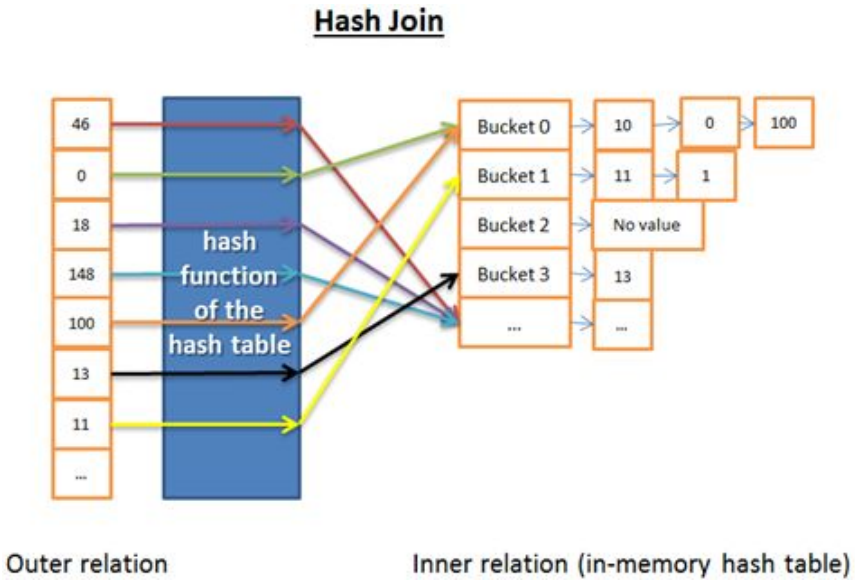
Joins

ITMO UNIVERSITY

ClickHouse uses **hash join** algorithm.
It takes **RIGHT** table/subquery and create a hash table in RAM.
That's why it's important put small table into right side.

You can't use hash join If right side have large size and can't fit in memory. In this case your query just fails due to out of memory exception. Also you can't use hash join if your join follows non-equal logic for instance using '>', '<' or like expression.

ClickHouse do not uses indexes to perform JOIN operations



ITMO more than a UNIVERSITY

https://makeomatic.ru/blog/2015/10/12/relational_database_2/

Joins: small hint

ITMO UNIVERSITY

Since ClickHouse do not uses indices to perform JOIN operations, sometimes it leads to long processing time.
For example:

```
SELECT p.*
FROM vk.posts AS p
INNER JOIN
(
    SELECT
        owner_id,
        item_id
    FROM vk.likes
    WHERE liker_id = 184921762
) AS l ON (l.owner_id = p.owner_id) AND (l.item_id = p.post_id)
INTO OUTFILE '/home/ncct/Desktop/_SHARED_DATA/select_posts.json'
FORMAT JSONEachRow


929 rows in set. Elapsed: 304.946 sec. Processed 1.92 billion rows, 1.38 TB (6.30 million rows/s., 4.54 GB/s.)
```

ITMO more than a UNIVERSITY

Query is completed in 305 second.

Joins: small hint

ITMO UNIVERSITY

Let’s look execution time for right subquery:

```
SELECT
    owner_id,
    item_id
FROM vk.likes
WHERE liker_id = 184921762
INTO OUTFILE '/home/ncct/Desktop/_SHARED_DATA/select_only_likes.json'
FORMAT JSONEachRow


934 rows in set. Elapsed: 0.115 sec. Processed 57.26 thousand rows, 1.37 MB (496.04 thousand rows/s., 11.91 MB/s.)
```

Time is ~0,1 sec. It means all the time ClickHouse tried to filter posts using full-scan.

ITMO more than a UNIVERSITY

Joins: small hint

ITMO UNIVERSITY

Since likes filtering is “cheap” query, we’ll try to prefilter posts by repeating this subquery in section IN (IN uses index).
Execution time of modified query:

2.3 sec vs. 305 sec

x130+ times faster

```
SELECT p.*
FROM
(
    SELECT *
    FROM vk.posts
    WHERE (owner_id, post_id) IN
    (
        SELECT
            owner_id,
            item_id AS post_id
        FROM vk.likes
        WHERE liker_id = 184921762
    )
) AS p
INNER JOIN
(
    SELECT
        owner_id,
        item_id
    FROM vk.likes
    WHERE liker_id = 184921762
) AS l ON (l.owner_id = p.owner_id) AND (l.item_id = p.post_id)
INTO OUTFILE '/home/ncct/Desktop/_SHARED_DATA/select_posts_query_used_index.json'
FORMAT JSONEachRow


929 rows in set. Elapsed: 2.302 sec. Processed 2.41 million rows, 33.55 MB (1.05 million rows/s., 14.57 MB/s.)
```

ITMOre than a UNIVERSITY

Joins: partial merge join

ITMO UNIVERSITY

If you'll have to join big tables (right and left parts cannot fit in memory), you may use another join algorithm: partial merge join.
It works only for LEFT and INNER joins.
This algorithm flushes intermediate data into hard drive if not enough ram.
You can activate it by following way:

```
SET partial_merge_join = 1
SET partial_merge_join_optimizations = 1
```

WARNING: partial merge join is slow operation.

ITMOre than a UNIVERSITY

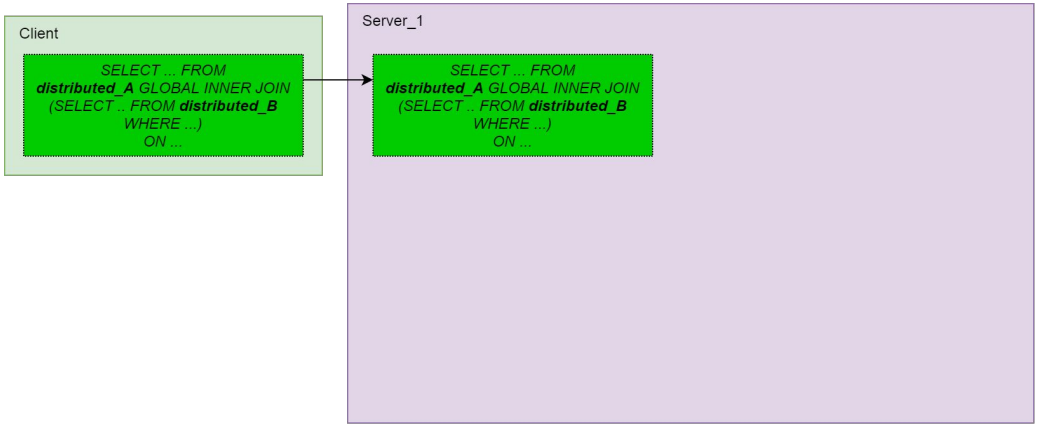
Distributed IN/JOIN

ITMO UNIVERSITY

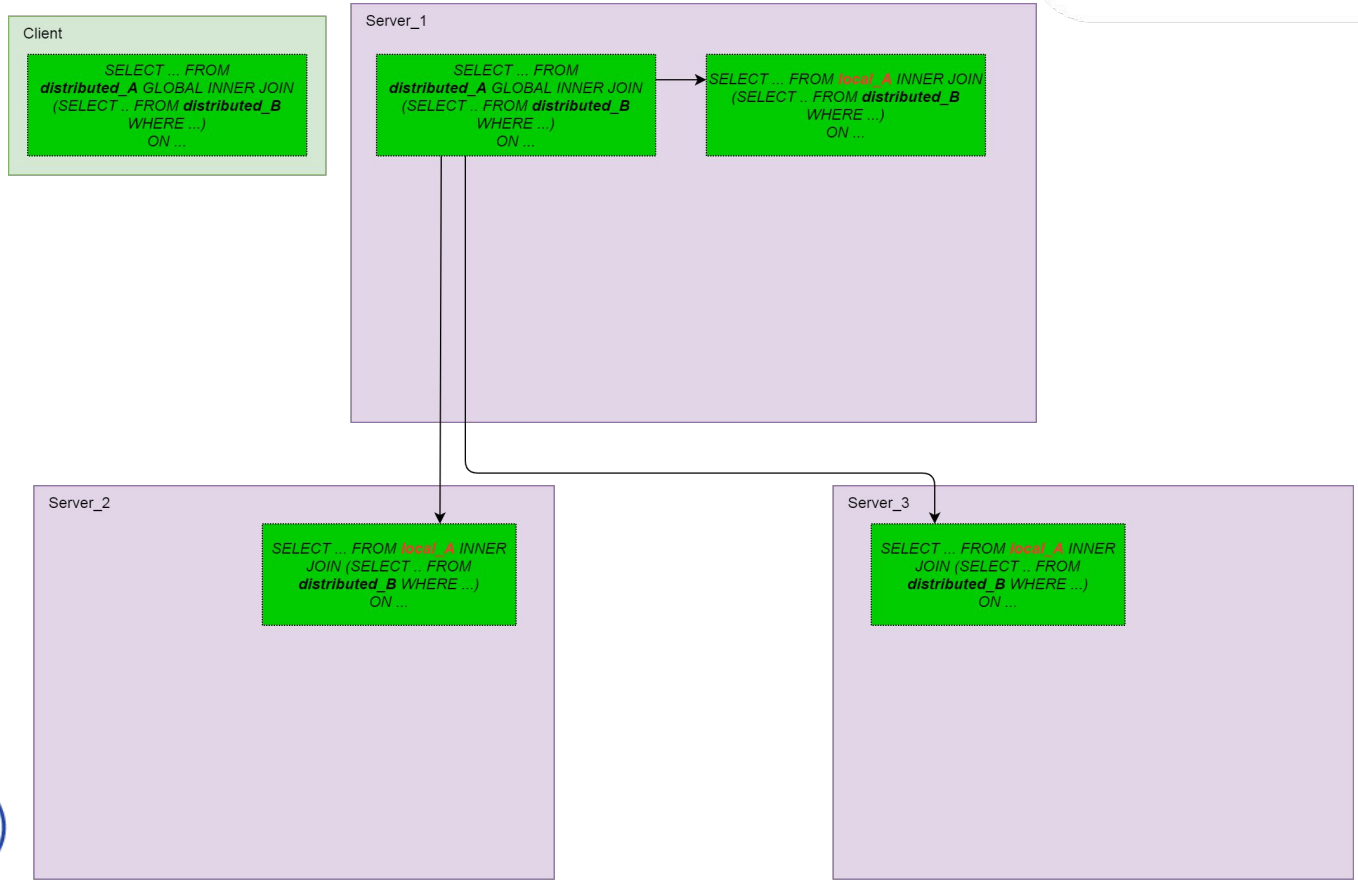
```
SELECT dl.user_id, dp.post_id, dp.text
FROM distributed_posts dp
INNER JOIN
   (SELECT user_id, post_id
    FROM distributed_likes
    WHERE user_id IN (123456, 456846, …)) dl
ON (dl.post_id = dp.post_id)

SET distributed_product_mode='allow'
```

ITMO more than a UNIVERSITY

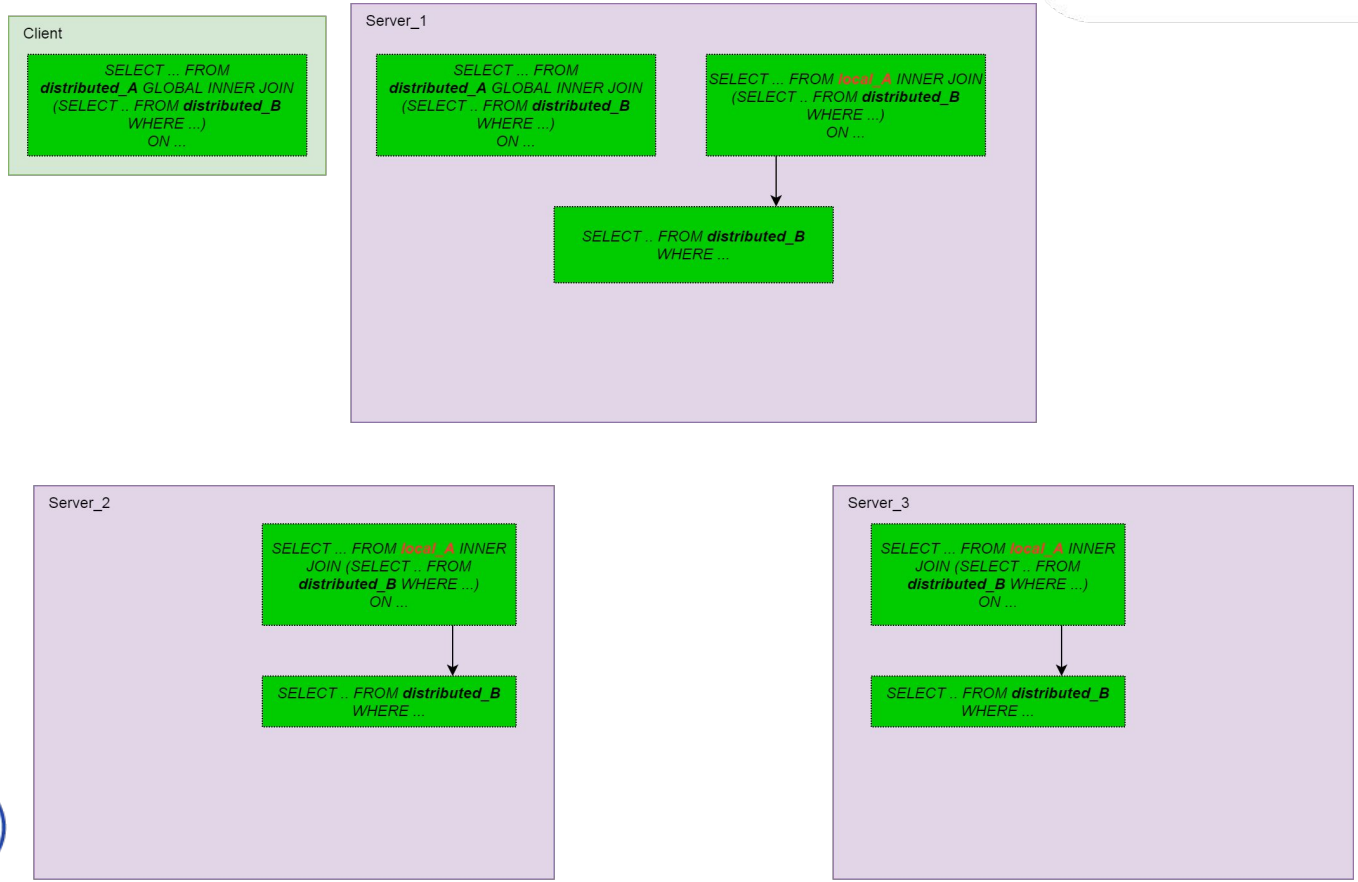
Distributed join: ALLOW



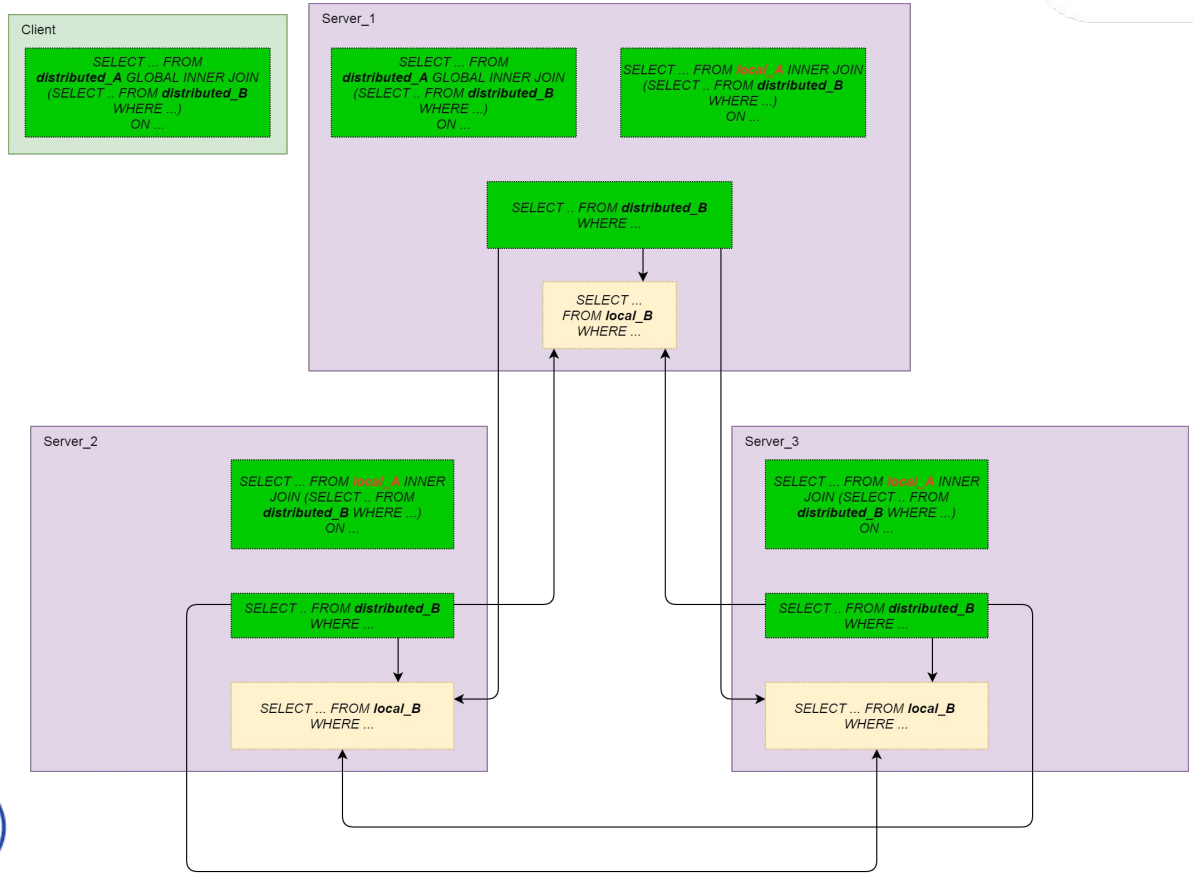
Distributed join: ALLOW



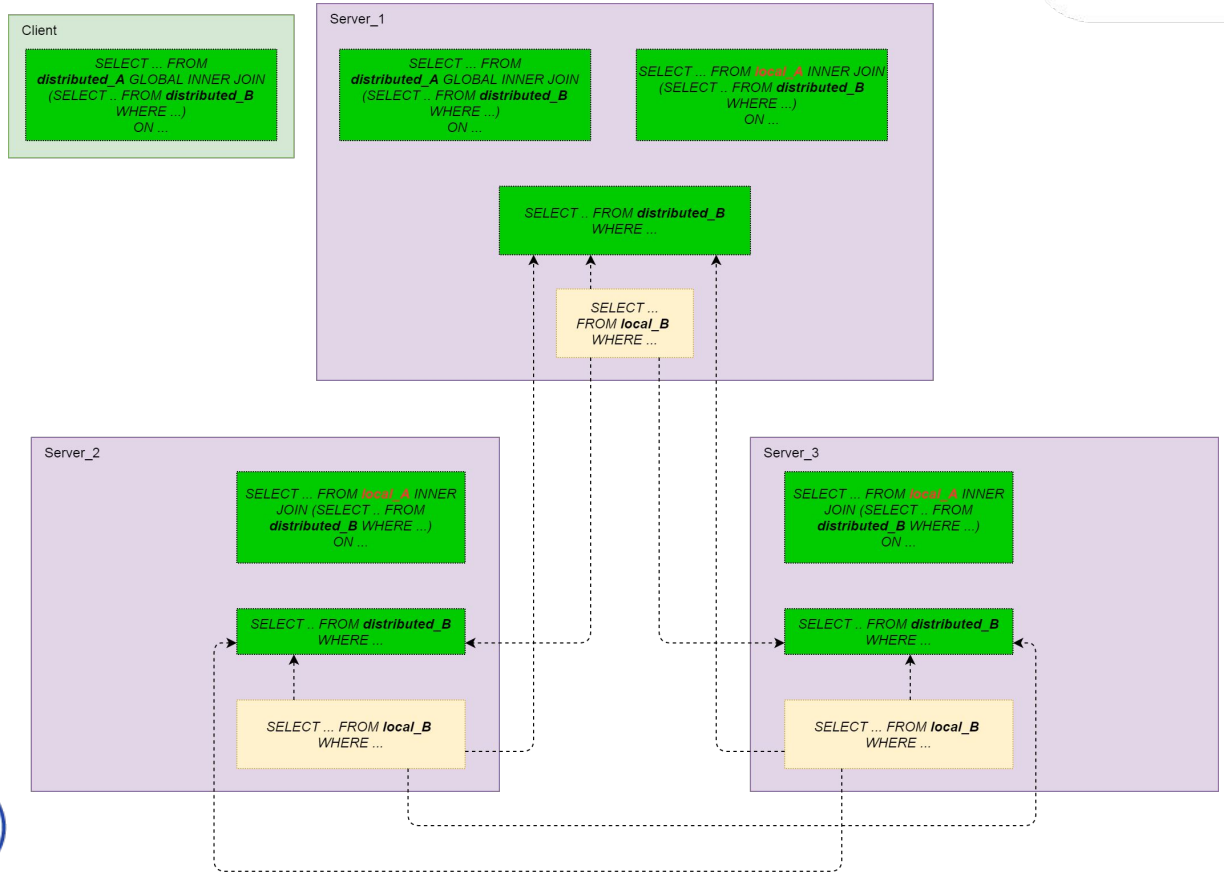
Distributed join: ALLOW



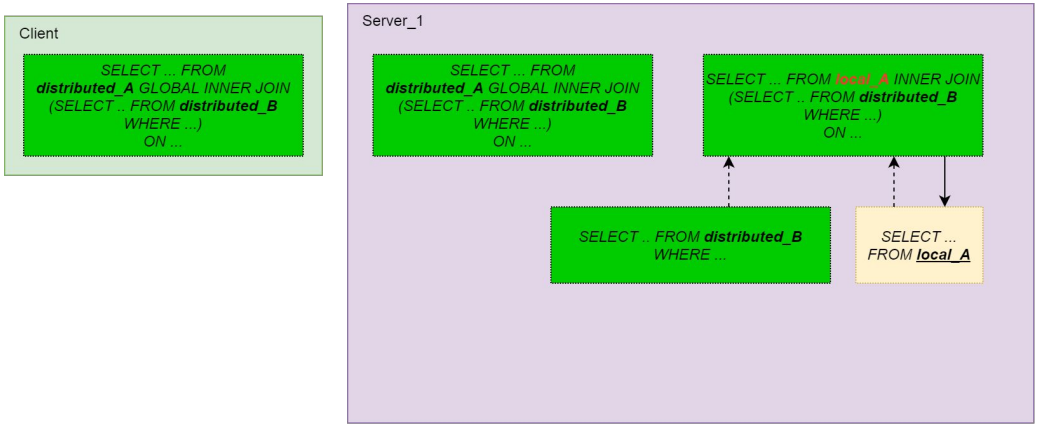
Distributed join: ALLOW

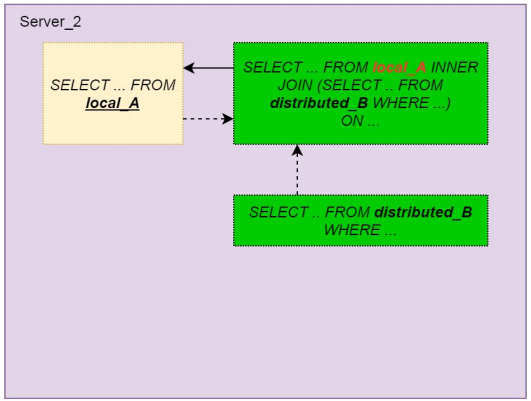


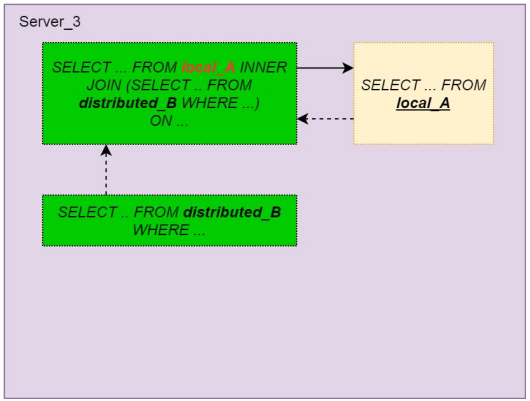
Distributed join: ALLOW



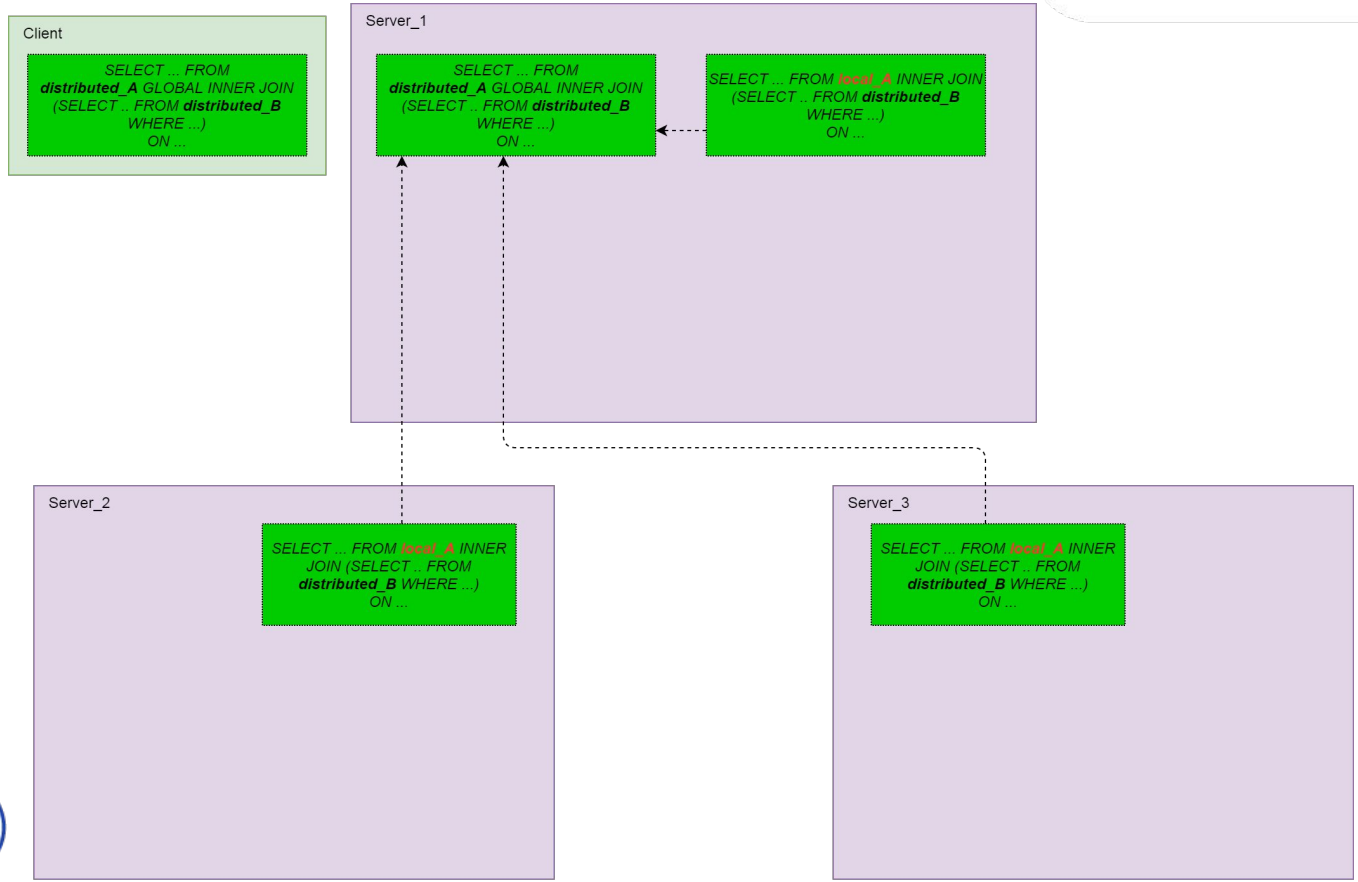
Distributed join: ALLOW



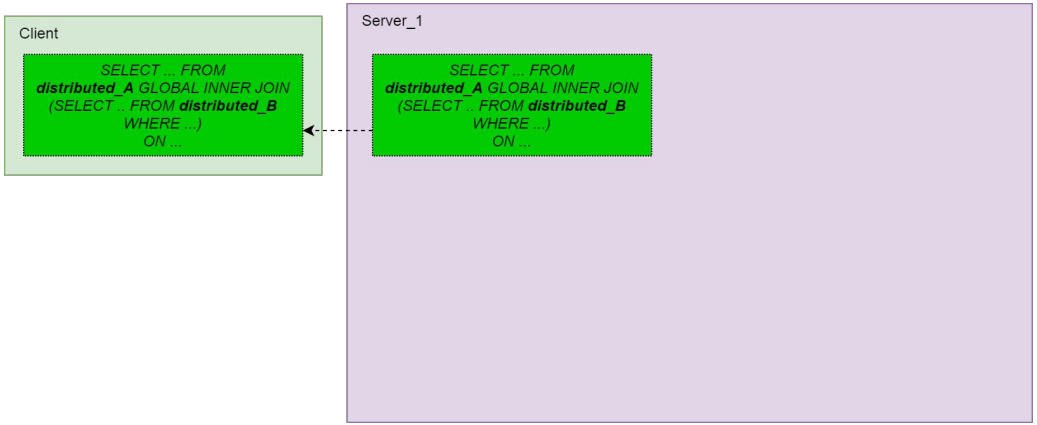




Distributed join: ALLOW



Distributed join: ALLOW



Distributed join: GLOBAL

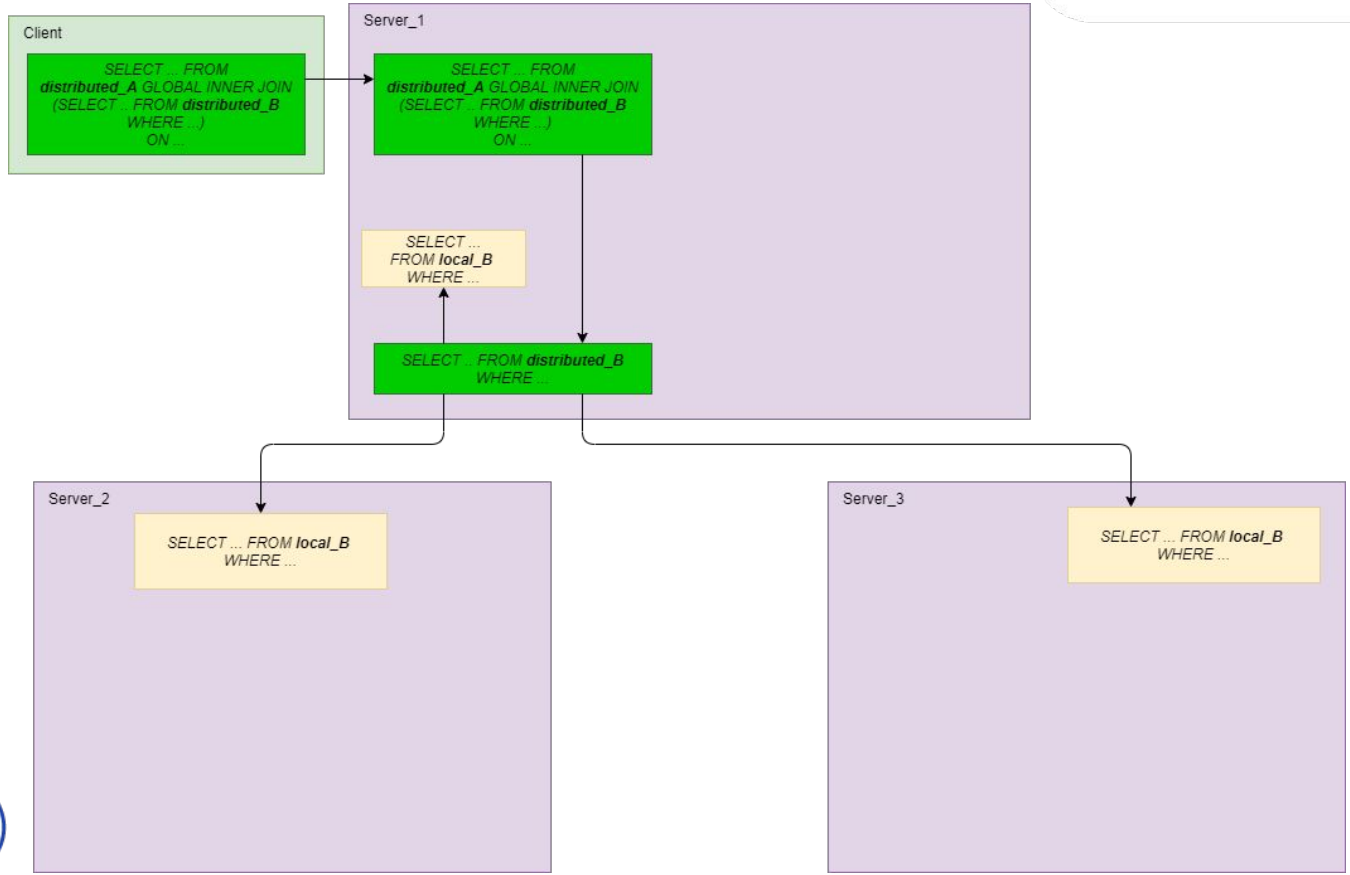
ITMO UNIVERSITY

```
SELECT dl.user_id, dp.post_id, dp.text
FROM distributed_posts dp
GLOBAL INNER JOIN
   (SELECT user_id, post_id
    FROM distributed_likes
    WHERE user_id IN (123456, 456846, …)) dl
ON (dl.post_id = dp.post_id)
```

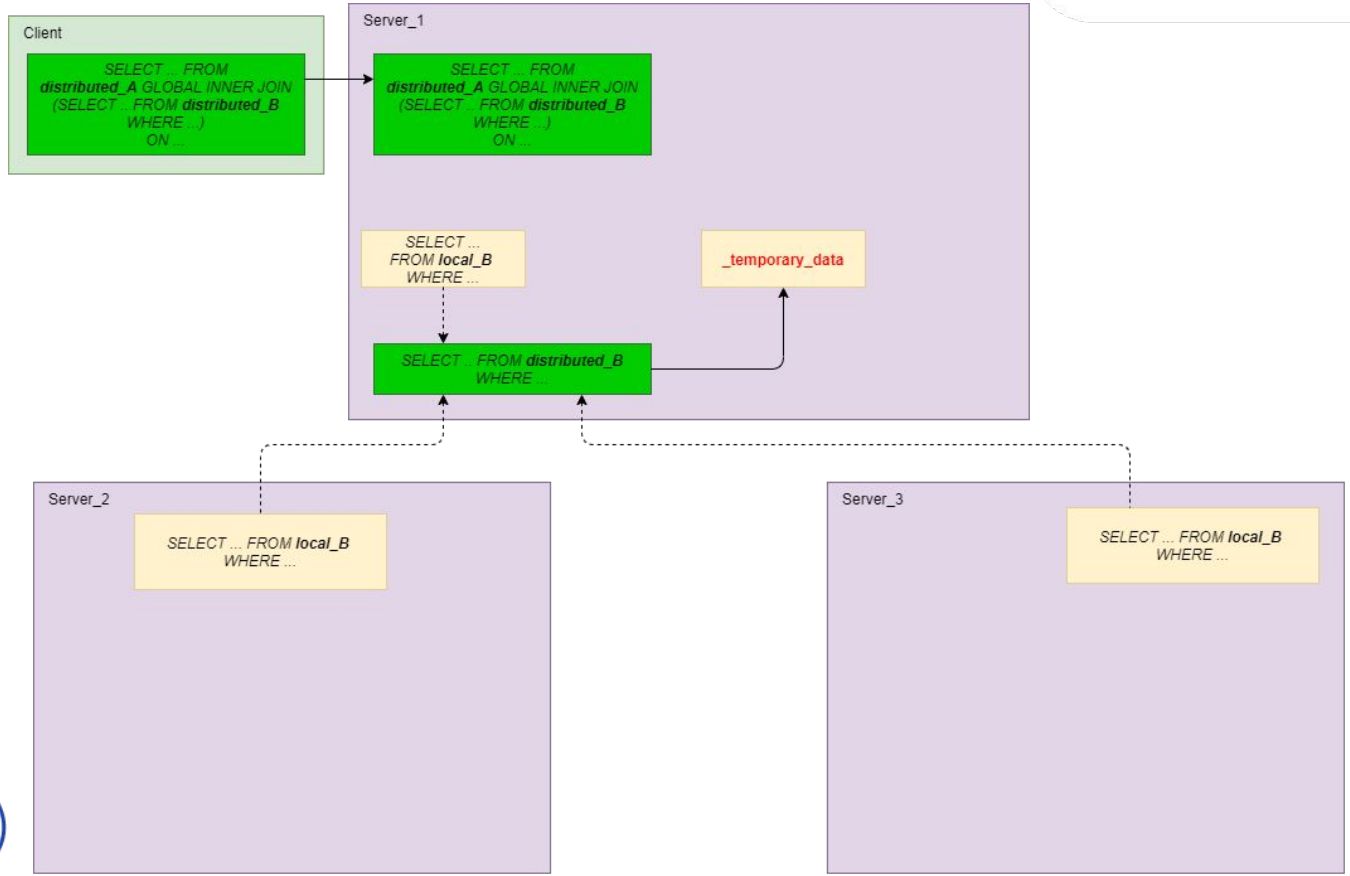
```
SET distributed_product_mode='global'
```

ITMO more than a UNIVERSITY

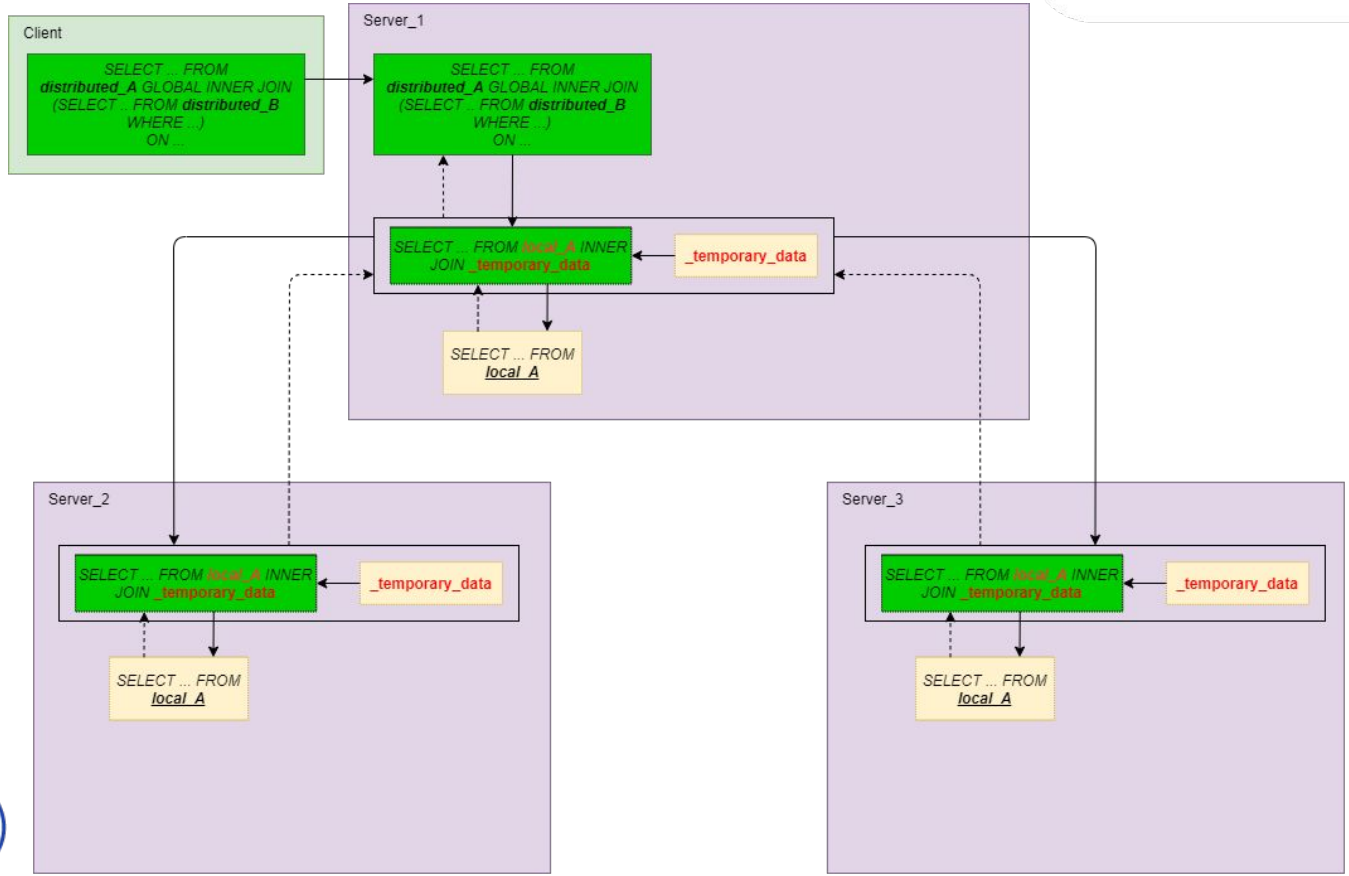
Distributed join: GLOBAL



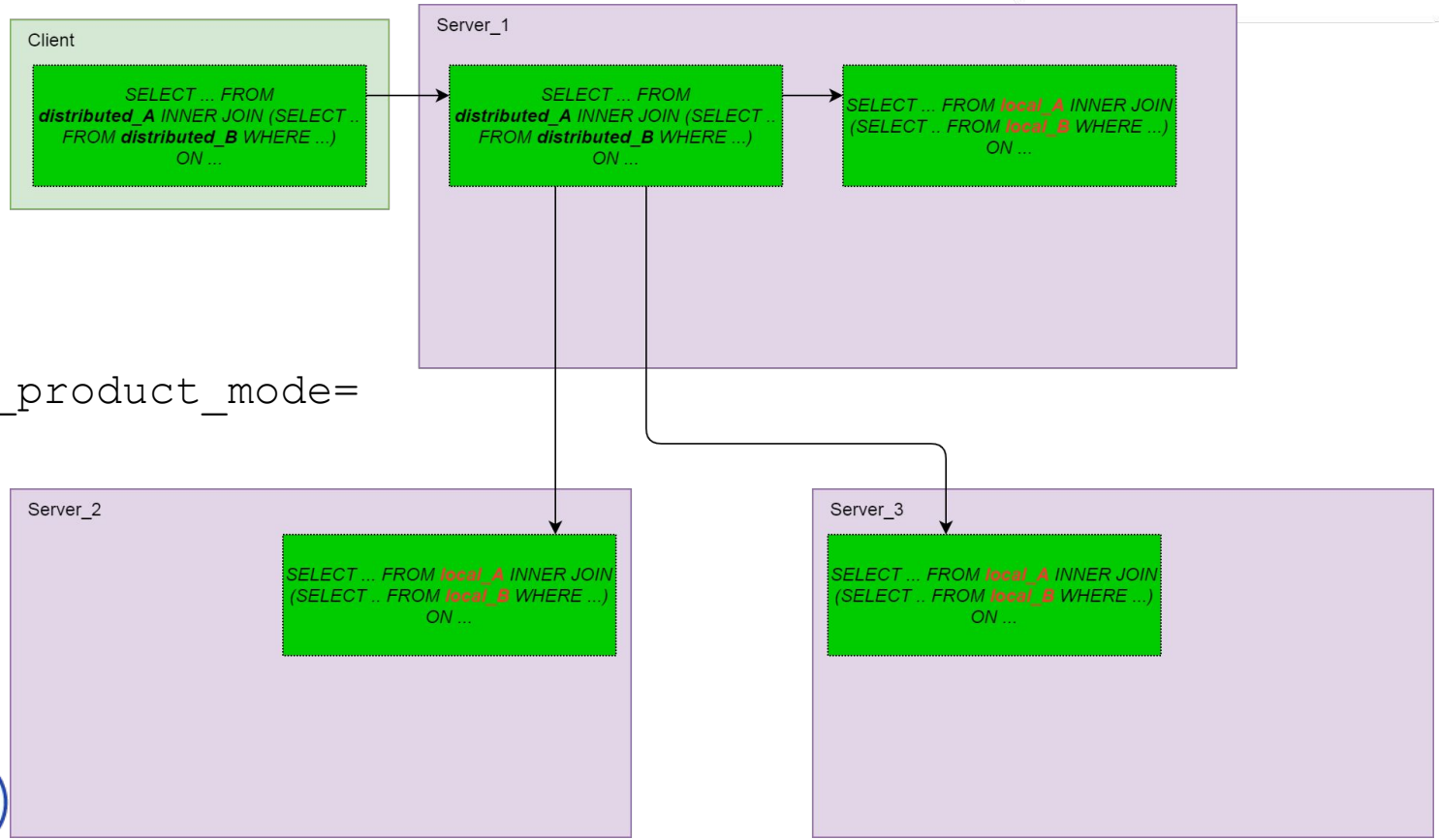
Distributed join: GLOBAL



Distributed join: GLOBAL

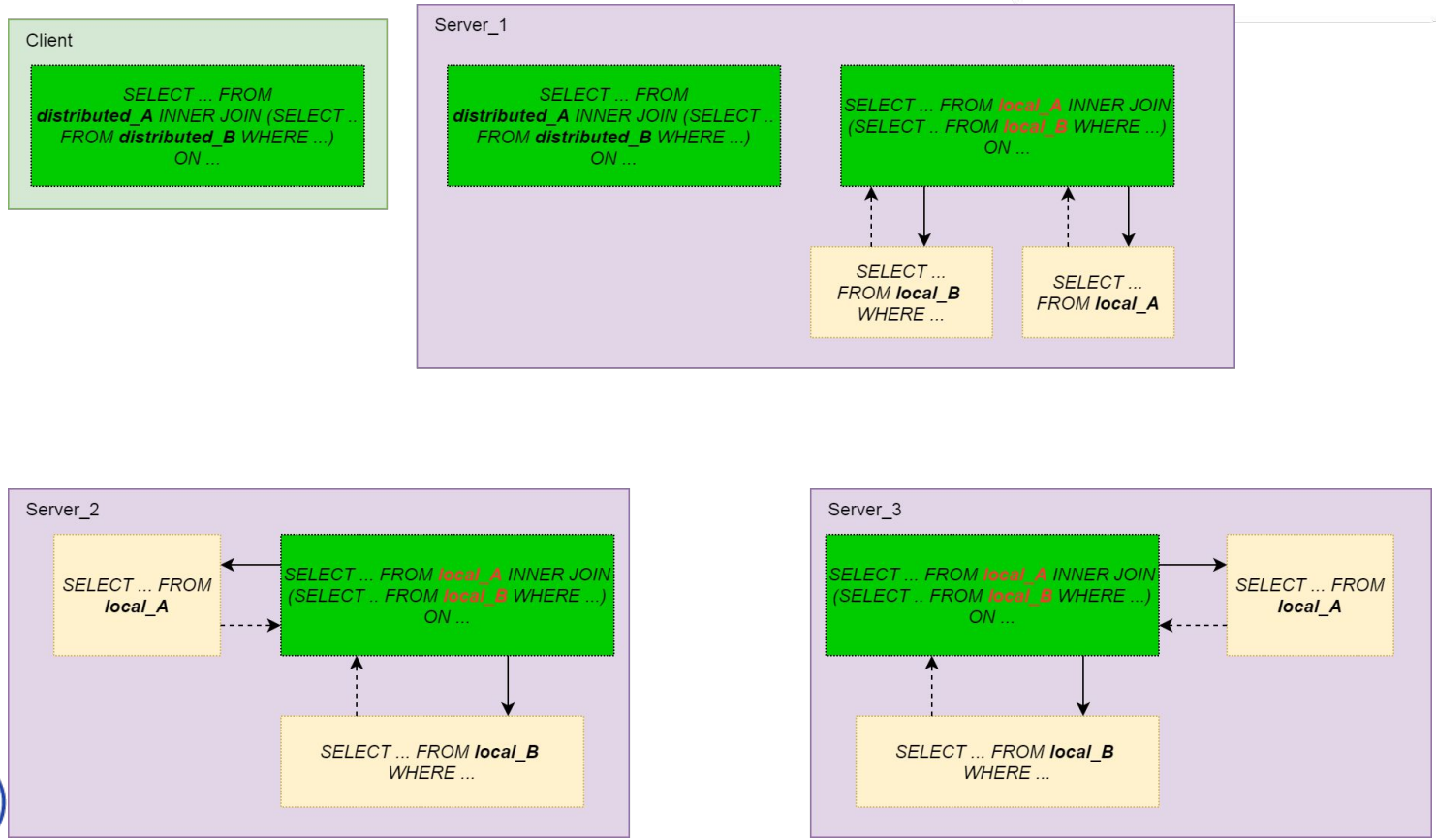


Distributed join: LOCAL

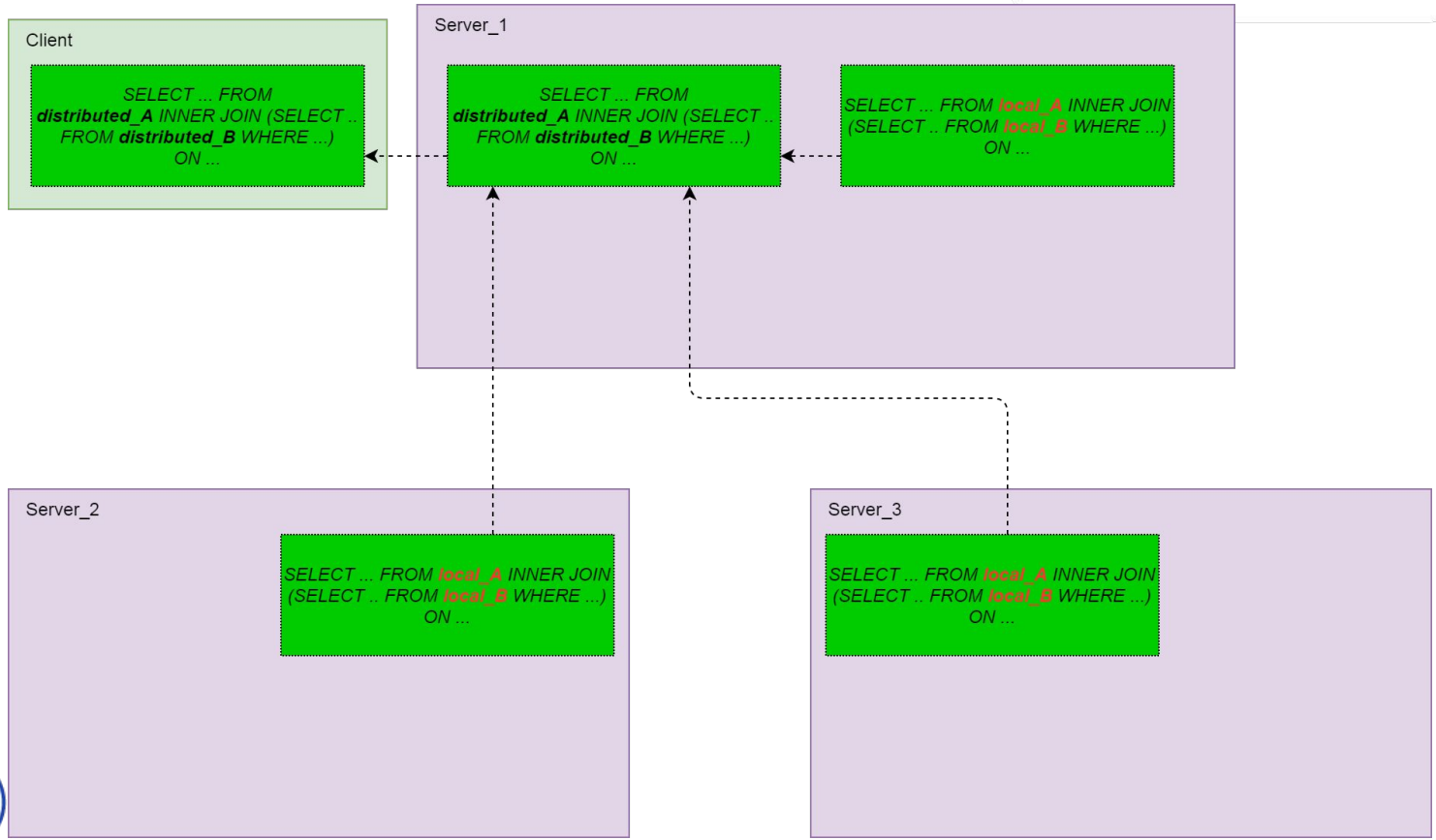



```
SET
distributed_product_mode=
'local'
```

Distributed join: LOCAL



Distributed join: LOCAL



Distributed join: LOCAL

ITMO UNIVERSITY

Requirements for ‘local’ mode usage:

- Data were sharded with same sharding key and expression
- IN/JOIN uses sharding key in section ‘ON’
- Data shuffle is unnecessary

Correct usage example:
posts table’s sharding key: xxhash(**owner_id**)
likes table’s sharding key: xxhash(**owner_id**)

```
SELECT … FROM distributed_posts p
LEFT JOIN (SELECT …
                FROM distributed_likes
                WHERE …) l
ON (p.owner_id = l.owner_id
AND l.item_id = p.post_id)
```

ITMO more than a UNIVERSITY

Distributed join: LOCAL

ITMO UNIVERSITY

Incorrect usage example:
posts table's sharding key: xxhash(**owner_id**)
likes table's sharding key: xxhash(**user_id**)

```
SELECT … FROM
distributed_posts p
LEFT JOIN
    (SELECT …
     FROM distributed_likes
     WHERE …) l
ON (p.owner_id = l.owner_id
AND l.item_id = p.post_id)
```

Another example:
posts table's sharding key: xxhash(**owner_id**)
likes table's sharding key: xxhash(**owner_id**)

```
SELECT … FROM
distributed_posts p
LEFT JOIN
(SELECT …
 FROM distributed_likes
 WHERE …) l
ON (p.owner_id = l.user_id
AND l.item_id = p.post_id)
```

ITMO more than a UNIVERSITY

Distributed join: LOCAL

ITMO UNIVERSITY

Another way to achieve data locality is providing replication of necessary tables.
Disadvantage of this approach is increased usage of hard drives space.

ITMOre than a UNIVERSITY

Joins: common recommendations

ITMO UNIVERSITY

- Since ClickHouse generally uses hash join algorithm, it takes **RIGHT** table/subquery and creates a hash table in RAM. That's why it's important put small table into right side.
- You should try to use indices when it's possible.
- Try to keep data locality. It allows you use 'local' distributed_product_mode.
- Be free in using prefilter to decrease size of processable table.

ITMOre than a UNIVERSITY

View: queries reusing

For example, we create view...

```
CREATE VIEW db.view AS
SELECT team, count(user) AS count
FROM db.users GROUP BY team;
```

...and reuse it in other queries. These queries are equals:

```
SELECT * FROM db.view
WHERE count > 2;
```

==

```
SELECT * FROM (
        SELECT team, count(user) AS count
        FROM db.users GROUP BY team)
WHERE count > 2;
```

Materialized View (MatView)

ITMO UNIVERSITY

- MatView is a completely separate *physical* table with its *own* engine
- MatView stores data transformed by the corresponding `SELECT` query
- ClickHouse materializes data *upon insertions* into the base table
- Stored pre-aggregated data may be used for:
  - keeping actual statistics;
  - subselecting a set of data for faster access;
  - faster filtering and index emulating to speed up retrieval from the base table.

ITMO more than a UNIVERSITY

MatView Example

ITMO UNIVERSITY

First of all we create the base table:

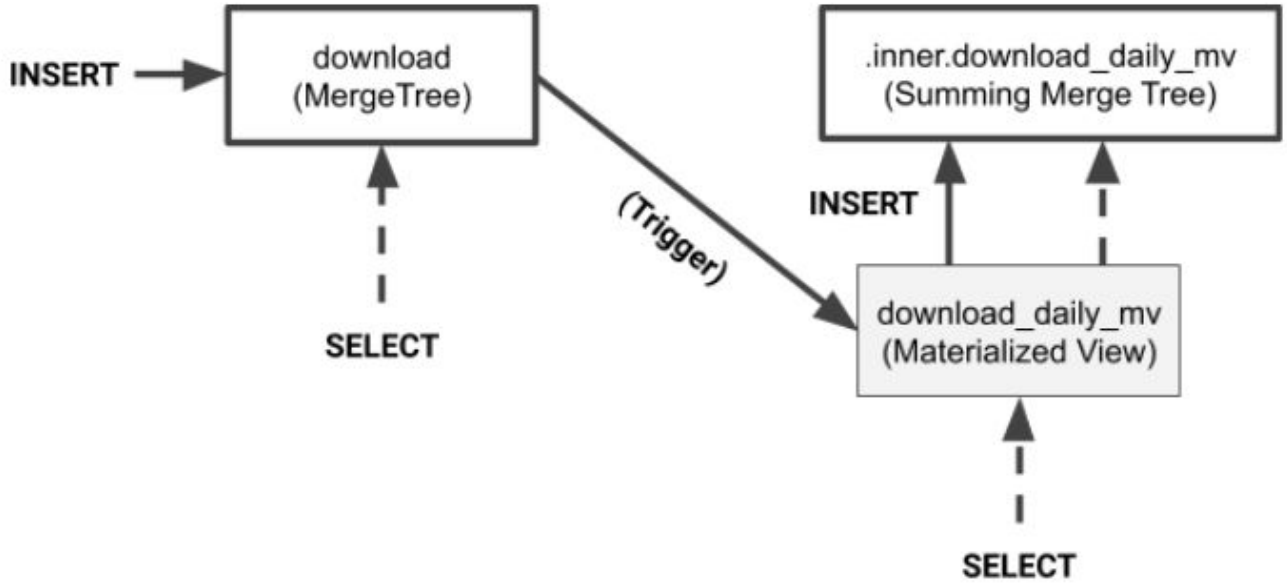
```
CREATE TABLE vk.likes
(
    item_type String,
    owner_id Int64,
    item_id Int64,
    liker_id Int64,
    ctime DateTime,
    like_date DateTime,
    post_date DateTime
)
ENGINE =
ReplacingMergeTree(ctime)
PARTITION BY toYYYYMM(post_date)
ORDER BY (liker_id, owner_id,
item_type, item_id);
```

And now we can create a matview:

```
CREATE MATERIALIZED VIEW
vk.matv_likes_count_per_month_by_liker_id
ENGINE = AggregatingMergeTree() PARTITION BY month
ORDER BY (liker_id, month)
POPULATE
AS SELECT
    liker_id,
    toYYYYMM(like_date) as month,
    countState(liker_id) as likes_count
FROM vk.likes
GROUP BY liker_id, toYYYYMM(like_date);
```

ITMO more than a UNIVERSITY

How MatView works?



https://www.altinity.com/blog/clickhouse-materialized-views-illuminated-part-1

Inserting data into ClickHouse

ITMO UNIVERSITY

- Many small inserts are bad -> too many parts are being created
  - slow SELECT queries;
  - many merges are required to optimize data in background.
- Large batch inserts are bad for clients
  - need to store data (it may require substantial amount of time to accumulate enough)
  - need to implement additional logic
  - multilang environments require even more work
- And still there may be just many clients... that doesn’t allow to address the problem to full extent

ITMO more than a UNIVERSITY

Inserting: Buffer Table

ITMO UNIVERSITY

Data flushed when:

- Match all the min* conditions
- Match at least one max* condition

Disadvantages:

- Data in different order and in different blocks
- Data may be lost
- No support of indexing

```
CREATE TABLE db.buffer_table
AS db.table
ENGINE = Buffer(db, /* database */
            table, /* table */
            16, /* num_layers (16 - default) */
            10, /* min_time, sec */
            100, /*max_time, sec */
            10000, /* min_rows */
            1000000, /* max_rows */
            10000000,  /* min_bytes */
            100000000) /* max_bytes */;
```

RAM usage = max_bytes * num_layers (example: 100M*16 = ~ 1.5 Gb RAM)

```
CREATE TABLE vk.buffer_likes AS vk.likes
ENGINE = Buffer(vk, distr_likes, 16, 10, 100, 10000, 1000000, 10000000, 100000000)

Ok.

0 rows in set. Elapsed: 0.067 sec.
```

ITMO more than a UNIVERSITY

Inserting: Proxy Servers

ITMO UNIVERSITY

1 chproxy

chproxy, is an http proxy and load balancer for ClickHouse database.

Features:

- Per-user routing and response caching.
- Flexible limits.
- Automatic SSL cerificate renewal.

Implemented in Go.

2 KittenHouse

KittenHouse is designed to be a local proxy between ClickHouse and application server in case it's impossible or inconvenient to buffer INSERT data on your application side.

Features:

- In-memory and on-disk data buffering.
- Per-table routing.
- Load-balancing and health checking.

Implemented in Go.

3 ClickHouse-Bulk

ClickHouse-Bulk is a simple ClickHouse insert collector.

Features:

- Group requests and send by threshold or interval.
- Multiple remote servers.
- Basic authentication.

Implemented in Go.

ITMOre than a UNIVERSITY

Integration table engines

ITMO UNIVERSITY

Why do we need them?: such engines allow to import or access data from external storages and process them using ClickHouse facilities

Still there is not many of them, but their numbers are growing

- Kafka
- MySQL
- JDBC
- ODBC
- HDFS

ITMO More than a UNIVERSITY

Kafka engine: a piece from streams world

ITMO UNIVERSITY

Kafka lets you:

- Publish or subscribe to data flows.
- Process streams as they become available and either storing them or keeping updated some representations upon events.
- Reliable buffering!

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2],
    ...
) ENGINE = Kafka()
SETTINGS
    kafka_broker_list = 'host:port',
    kafka_topic_list = 'topic1,topic2,...',
    kafka_group_name = 'group_name',
    kafka_format = 'data_format'[,]
    [kafka_row_delimiter = 'delimiter_symbol',]
    [kafka_schema = '',]
    [kafka_num_consumers = N,]
    [kafka_skip_broken_messages = N]
```

The delivered messages are tracked automatically, so each message in a group is counted *only once*.

ITMOre than a UNIVERSITY

ENGINE = Kafka()

ITMO UNIVERSITY

SELECT is not particularly useful for reading messages (except for debugging), because each message can be read only once. It is more practical to create real-time threads using materialized views. To do this:

1. Use the engine to create a Kafka consumer and consider it a data stream.
2. Create a table with the desired structure.
3. Create a materialized view that converts data from the engine and puts it into a previously created table.

https://clickhouse.yandex/docs/en/operations/table_engines/kafka/

ITMO more than a UNIVERSITY

Example: Kafka engine + MV

ITMO UNIVERSITY

```
CREATE TABLE queue (
  timestamp UInt64,
  level String,
  message String
) ENGINE = Kafka('localhost:9092', 'topic', 'group1', 'JSONEachRow');

CREATE TABLE daily (
  day Date,
  level String,
  total UInt64
) ENGINE = SummingMergeTree(day, (day, level), 8192);

CREATE MATERIALIZED VIEW consumer TO daily
  AS SELECT toDate(toDateTime(timestamp)) AS day, level, count() as total
  FROM queue GROUP BY day, level;

SELECT level, sum(total) FROM daily GROUP BY level;
```

To improve performance, received messages are grouped into blocks the size of `max_insert_block_size`. If the block wasn't formed within `stream_flush_interval_ms` milliseconds, the data will be flushed to the table regardless of the completeness of the block.

To stop receiving topic data or to change the conversion logic, detach the materialized view:

```
DETACH TABLE consumer;
ATTACH MATERIALIZED VIEW consumer;
```

ITMOre than a UNIVERSITY

Conclusion

ITMO UNIVERSITY

- ClickHouse is a powerful MPP database that provides rich capabilities for data modelling and efficient data retrieval

- Main concepts and mechanisms of ClickHouse has been presented.

ITMOre than a UNIVERSITY

Thanks for your attention!

ITMOre than a UNIVERSITY