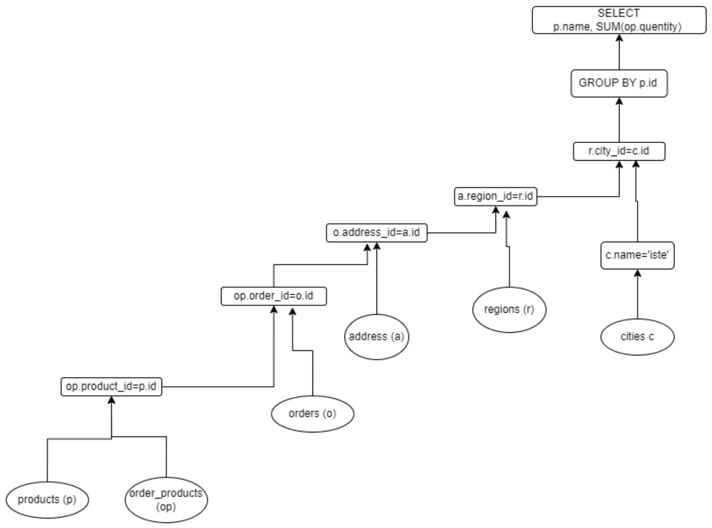
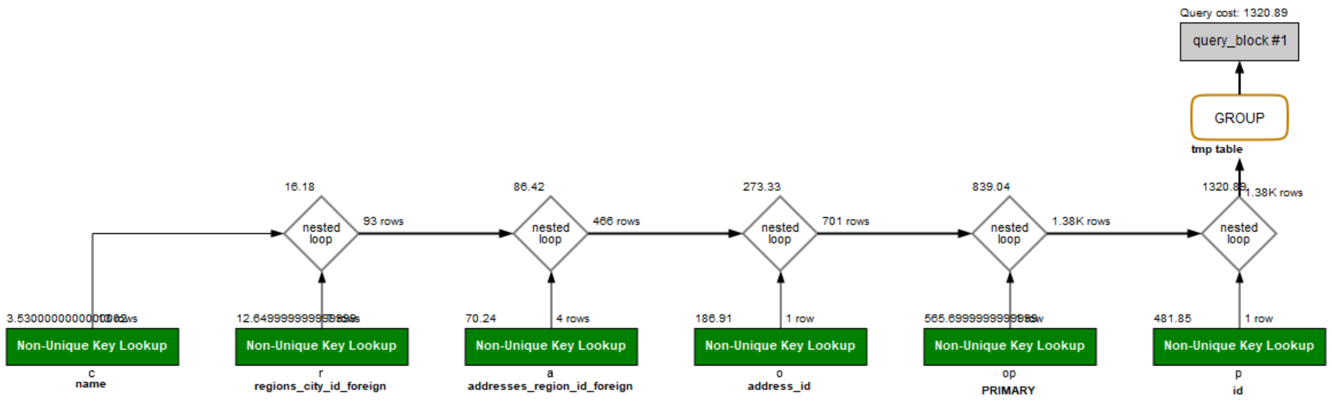
QUERY 1

get name of the products and its quantity ordered to addresses in (iste) city

```
select p.name,SUM(op.quentity) as order_times
from products p
join order_products op
  on op.product_id=p.id
join orders o
  on op.order_id=o.id
join addresses a
  on o.address_id=a.id
JOIN regions r
 ON a.region_id=r.id
JOIN cities c
 ON r.city_id=c.id
where c.name='iste'
GROUP BY p.id;
```

Query took 3.9464 seconds.

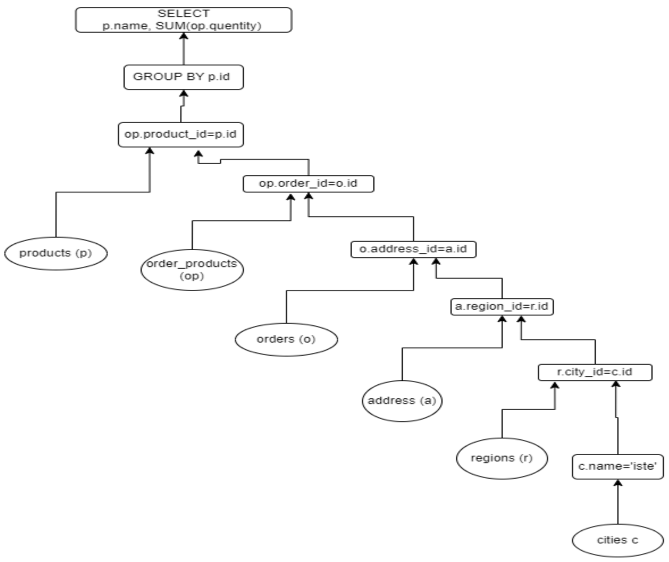




Optimization QUERY 1

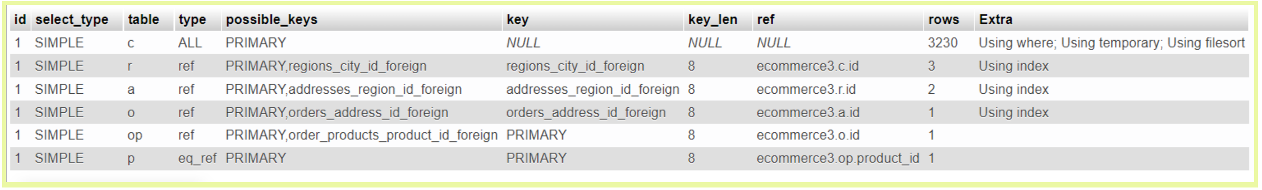
```
select p.name,SUM(op.quentity) as order_times
From cities c
JOIN regions r
ON r.city_id=c.id
join addresses a
 ON a.region_id=r.id
join orders o
  on o.address_id=a.id
join order_products op
  on op.order_id=o.id
Join products p
ON op.product_id=p.id
where c.name='iste'
GROUP BY p.id;
```

Query took 3.5253 seconds. ** almost the same time as the DBMS do its own optimization in ordering the join



In the optimization we rearranged the leaf nodes of the tree so that the leaf node relations with the most restrictive select operation are executed first in the query tree representation

Both query has the same execution plan in the DBMS



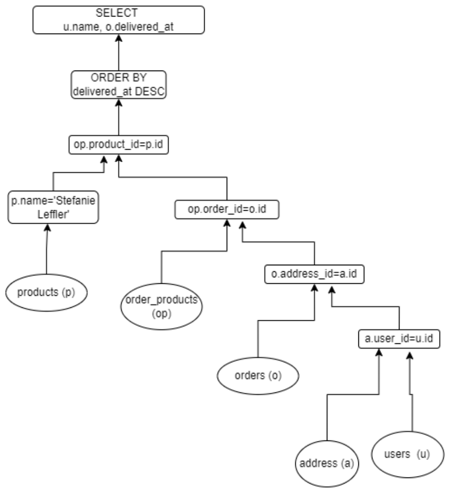
id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	c	ALL	PRIMARY	*NULL*	*NULL*	*NULL*	3230	Using where; Using temporary; Using filesort
1	SIMPLE	r	ref	PRIMARY,regions_city_id_foreign	regions_city_id_foreign	8	ecommerce3.c.id	3	Using index
1	SIMPLE	a	ref	PRIMARY,addresses_region_id_foreign	addresses_region_id_foreign	8	ecommerce3.r.id	2	Using index
1	SIMPLE	o	ref	PRIMARY,orders_address_id_foreign	orders_address_id_foreign	8	ecommerce3.a.id	1	Using index
1	SIMPLE	op	ref	PRIMARY,order_products_product_id_foreign	PRIMARY	8	ecommerce3.o.id	1	
1	SIMPLE	p	eq_ref	PRIMARY	PRIMARY	8	ecommerce3.op.product_id	1	

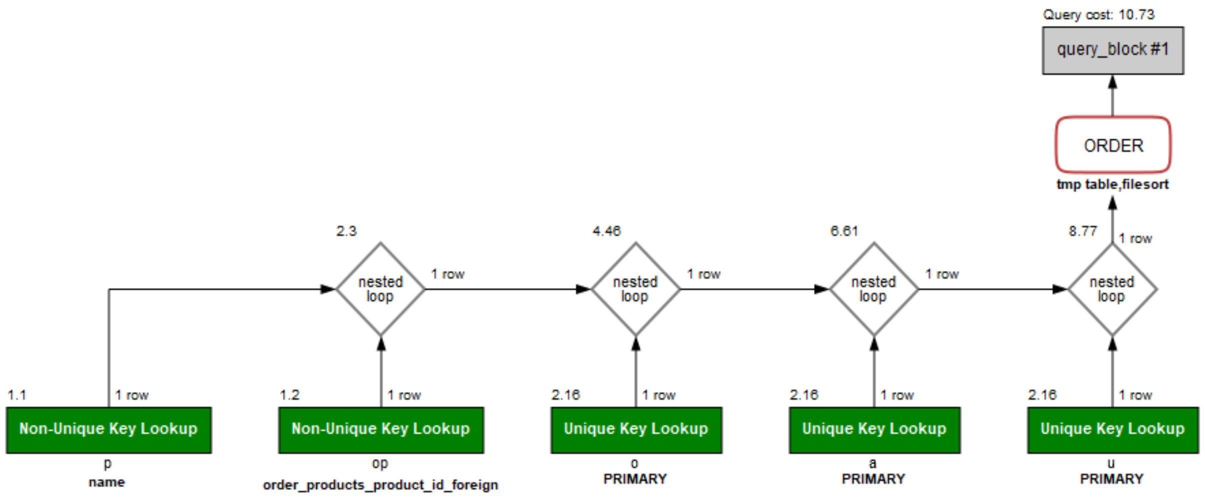
QUERY 2

get name of users and date of deliver they order specific product

```
select u.first_name,u.last_name, o.delivered_at
from users u
join addresses a
ON a.user_id=u.id
join orders o
on o.address_id=a.id
join order_products op
on op.order_id=o.id
join products p
on op.product_id=p.id
where p.name='Stefanie Leffler'
ORDER BY delivered_at DESC
```

Query took 0.437 seconds

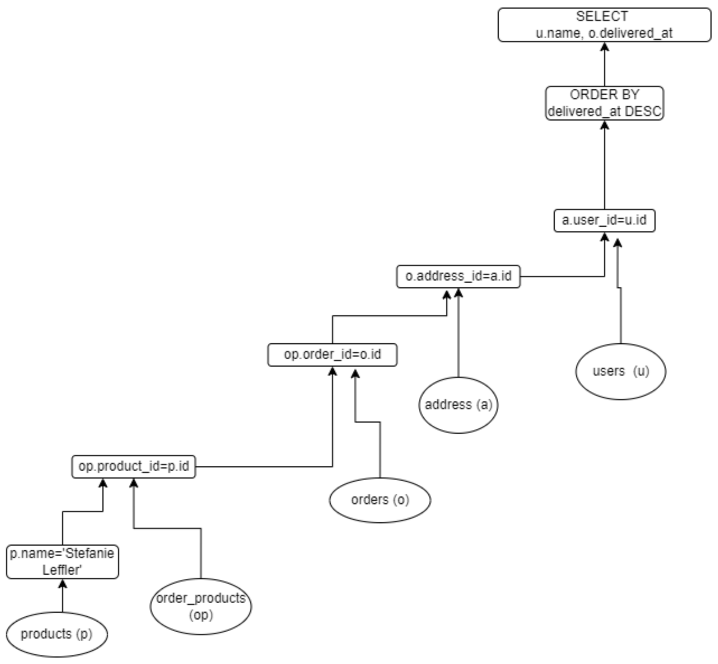




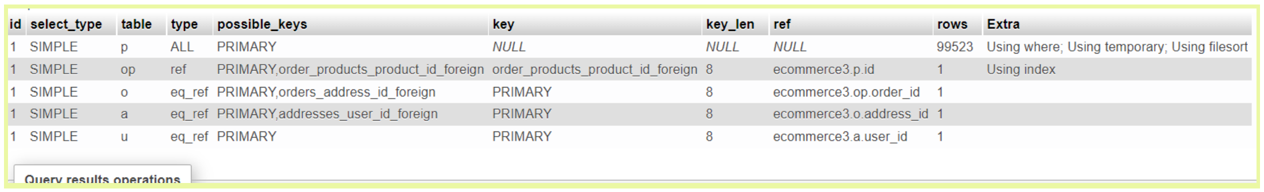
Optimization QUERY 2

```
select u.first_name,u.last_name, o.delivered_at
from products p
join order_products op
  on op.product_id=p.id
join orders o
  on op.order_id=o.id
join addresses a
  on o.address_id=a.id
JOIN users u
 ON a.user_id=u.id
where p.name='plapla8'
ORDER BY delivered_at DESC
```

Query took 0.506 seconds.)



In the optimization we rearranged the leaf nodes of the tree so that the leaf node relations with the most restrictive select operation are executed first in the query tree representation
Both query has the same execution plan in the DBMS



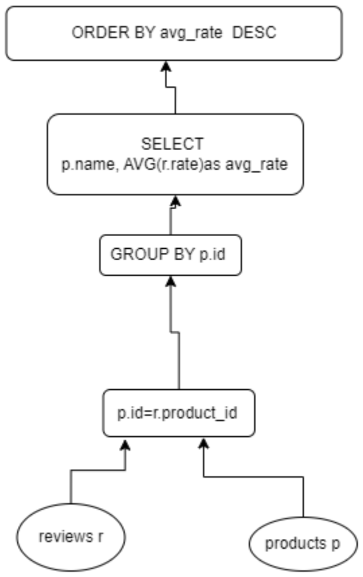
id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	p	ALL	PRIMARY	*NULL*	*NULL*	*NULL*	99523	Using where; Using temporary; Using filesort
1	SIMPLE	op	ref	PRIMARY,order_products_product_id_foreign	order_products_product_id_foreign	8	ecommerce3.p.id	1	Using index
1	SIMPLE	o	eq_ref	PRIMARY,orders_address_id_foreign	PRIMARY	8	ecommerce3.op.order_id	1	
1	SIMPLE	a	eq_ref	PRIMARY,addresses_user_id_foreign	PRIMARY	8	ecommerce3.o.address_id	1	
1	SIMPLE	u	eq_ref	PRIMARY	PRIMARY	8	ecommerce3.a.user_id	1	

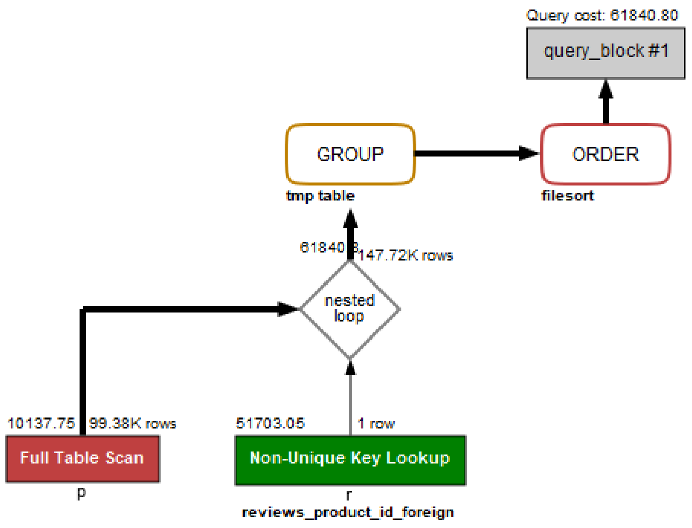
QUERY 3

get name of the products with that have rate with its average rate ordered by average rate

```
select p.name, AVG(r.rate)as avg_rate
from products p
join reviews r
on p.id=r.product_id
GROUP BY(p.id)
ORDER BY avg_rate DESC;
```

Query took 2.1291 seconds.





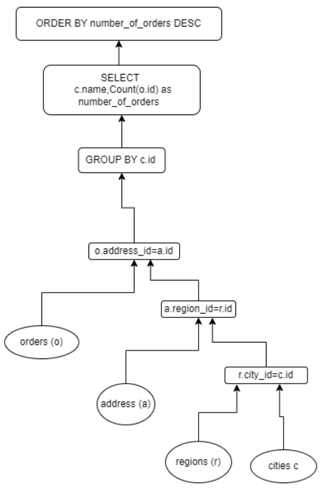
The outer join table is the small table product and the inner table is the large table review this query already optimized

QUERY 4

Get all cities ordered by number of orders to the address in the city

```
select c.name,Count(o.id) as number_of_orders
from orders o
 join addresses a
on o.address_id=a.id
JOIN regions r
ON a.region_id=r.id
JOIN cities c
ON r.city_id=c.id
GROUP BY c.id
ORDER BY number_of_orders DESC;
```

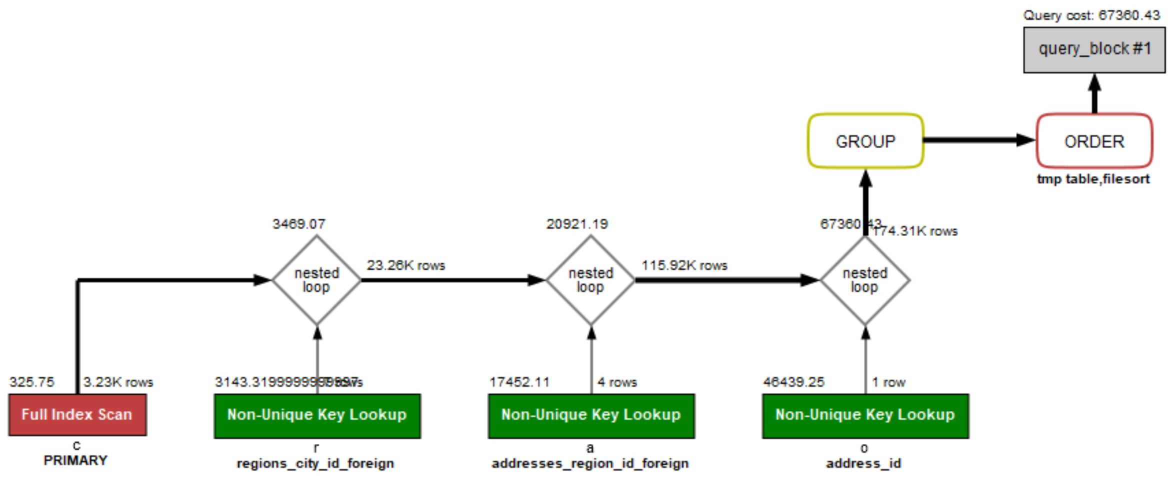
Query took 9.8346 seconds



Optimization QUERY 4

```
select c.name,Count(o.id) as number_of_orders
From cities c
JOIN regions r
ON r.city_id=c.id
JOIN addresses a
ON a.region_id=r.id
JOIN orders o
on o.address_id=a.id
GROUP BY c.id
ORDER BY number_of_orders DESC;
```

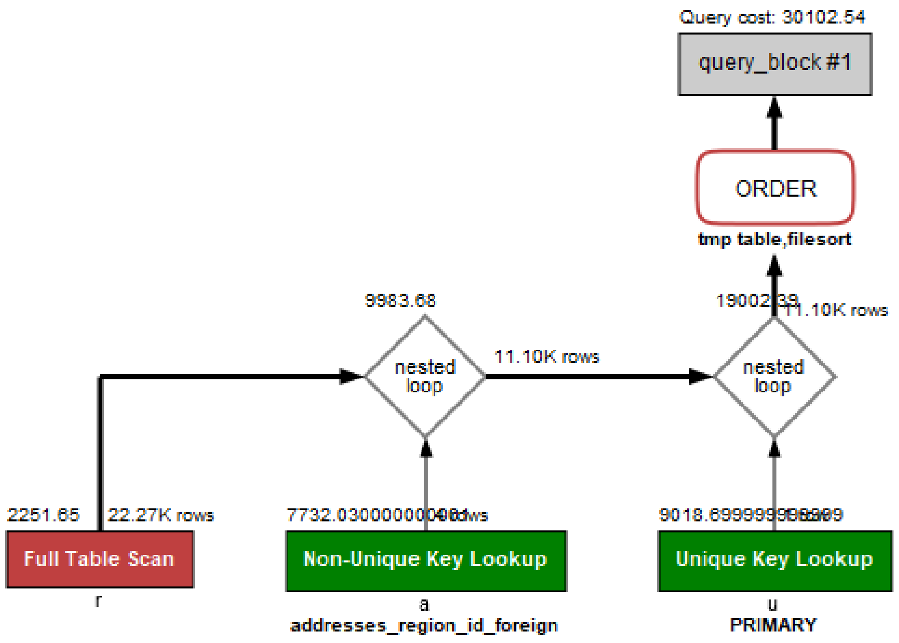
In the optimization we rearranged the leaf nodes of the tree so that the leaf node relations with the most restrictive select operation are executed first in the query tree representation



QUERY 5

Get all users ordered orders to address in region (eaque)

```
select u.first_name,u.last_name, a.street,a.building,a.floor
FROM users u
Join addresses a
ON a.user_id=u.id
JOIN regions r
 on r.id=a.region_id
where r.name="eaque"
ORDER BY  a.street
```
Query took 3.7409 seconds



Optimization QUERY 5

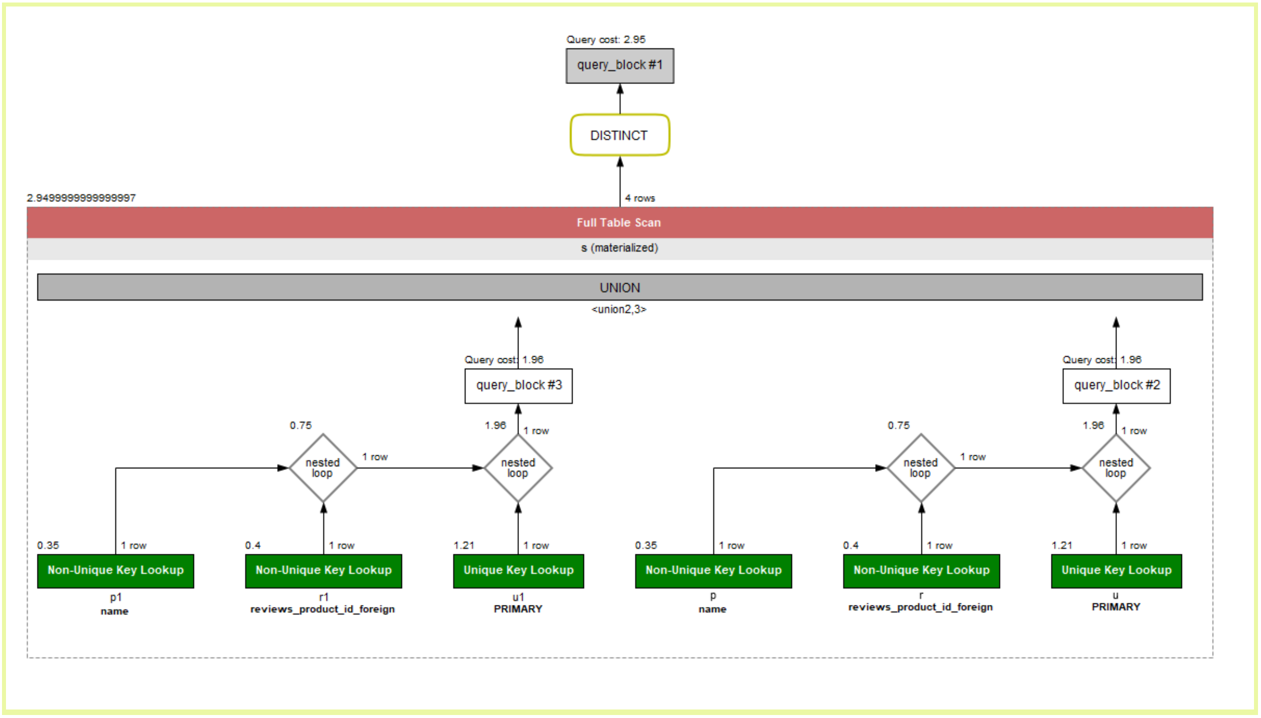
```
select u.first_name,u.last_name, a.street,a.building,a.floor
From regions r
join addresses a
 on r.id=a.region_id
JOIN users u
 ON a.user_id=u.id
where r.name="eaque"
ORDER BY  a.street
```
Query took 3.3691 seconds

QUERY 6

Get all users review product “plapla1” or “plapla2”

```
select DISTINCT(s.id),s.first_name,s.last_name
FROM
(select u.id,u.first_name,u.last_name
From products p
JOIN reviews r
 ON r.product_id=p.id
 JOIN users u
 ON u.id=r.user_id
 WHERE p.name="plapla5"
 UNION
 select u1.id,u1.first_name,u1.last_name
From products p1
JOIN reviews r1
 ON r1.product_id=p1.id
 JOIN users u1
 ON u1.id=r1.user_id
 WHERE p1.name="plapla2"
) as s;
```

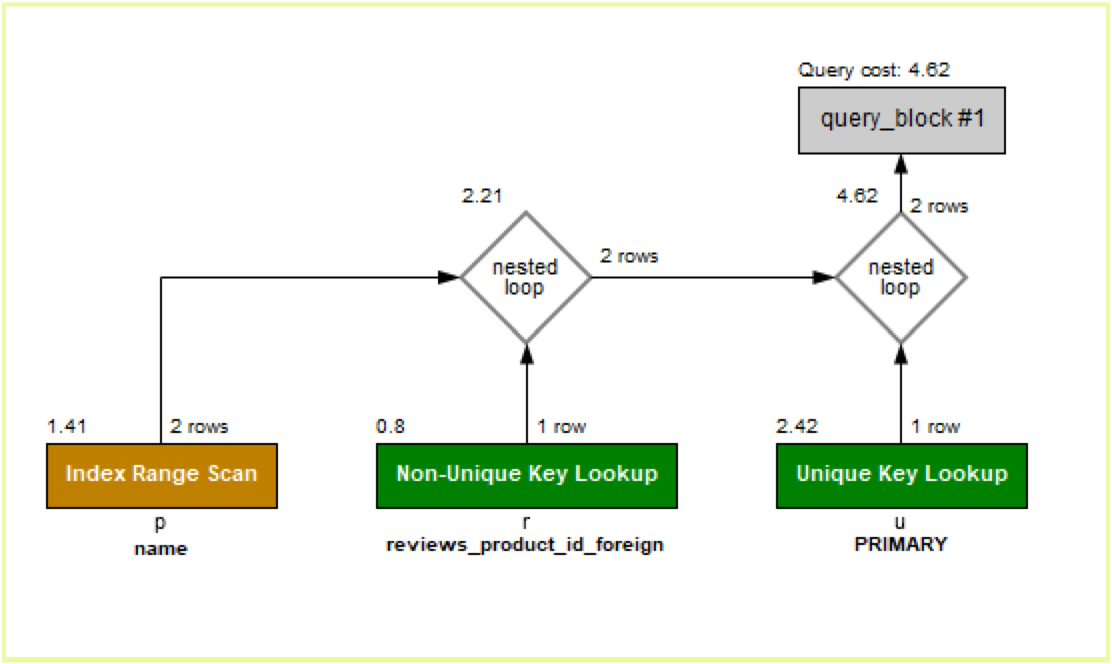
Query took 0.1533 seconds.



Optimization QUERY 6

```
select u.id,u.first_name,u.last_name
From products p
JOIN reviews r
ON r.product_id=p.id
JOIN users u ON
u.id=r.user_id
WHERE p.name="plapla5"
or p.name="plapla2";
```

Query took 0.0070 seconds



Changed the query instead of join the result from 2 query run on the same tables we run one query and change the where conditions

Time enhancement after query optimization

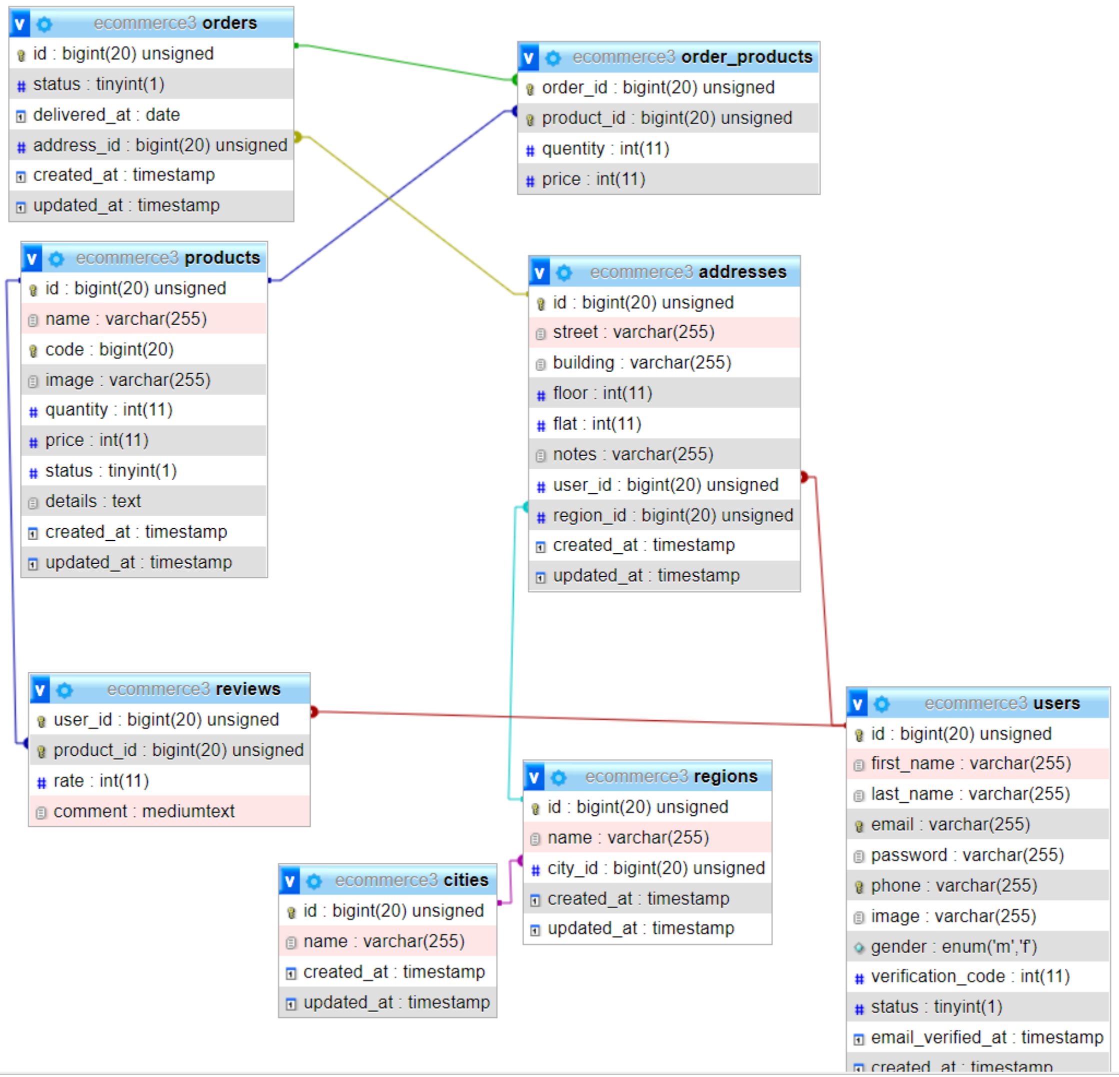
	Before query optimization	After query optimization	Percentage of time enhancement
Q1	Query took 3.9464 seconds.	Query took 3.5253 seconds.	10.67%
Q2	Query took 0.437 seconds	Query took 0.506 seconds.	-15.78%
Q3	Query took 2.1291 seconds.	–	–
Q4	Query took 9.8346 seconds	Query took 8.6039 seconds.	12.5139%
Q5	Query took 3.7409 seconds	Query took 3.3691 seconds	9.938%
Q6	Query took 0.1533 seconds.	Query took 0.0070 seconds	95.433%

Space enhancement after query optimization

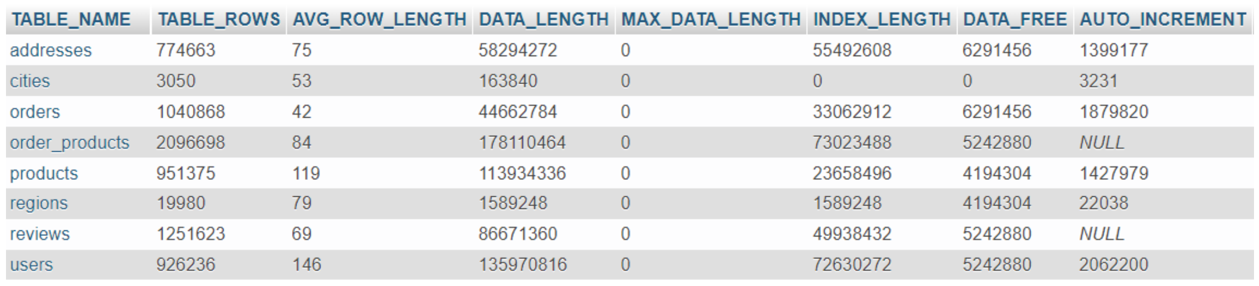
# rows processed	Before query optimization	After query optimization	Percentage of space enhancement
Q1	2725	2725	0%
Q2	0	11	-
Q3	349007	–	–
Q4	288388	288388	0%
Q5	23321	23321	0%
Q6	2	6	-200%

Schema optimization

1. Old schema

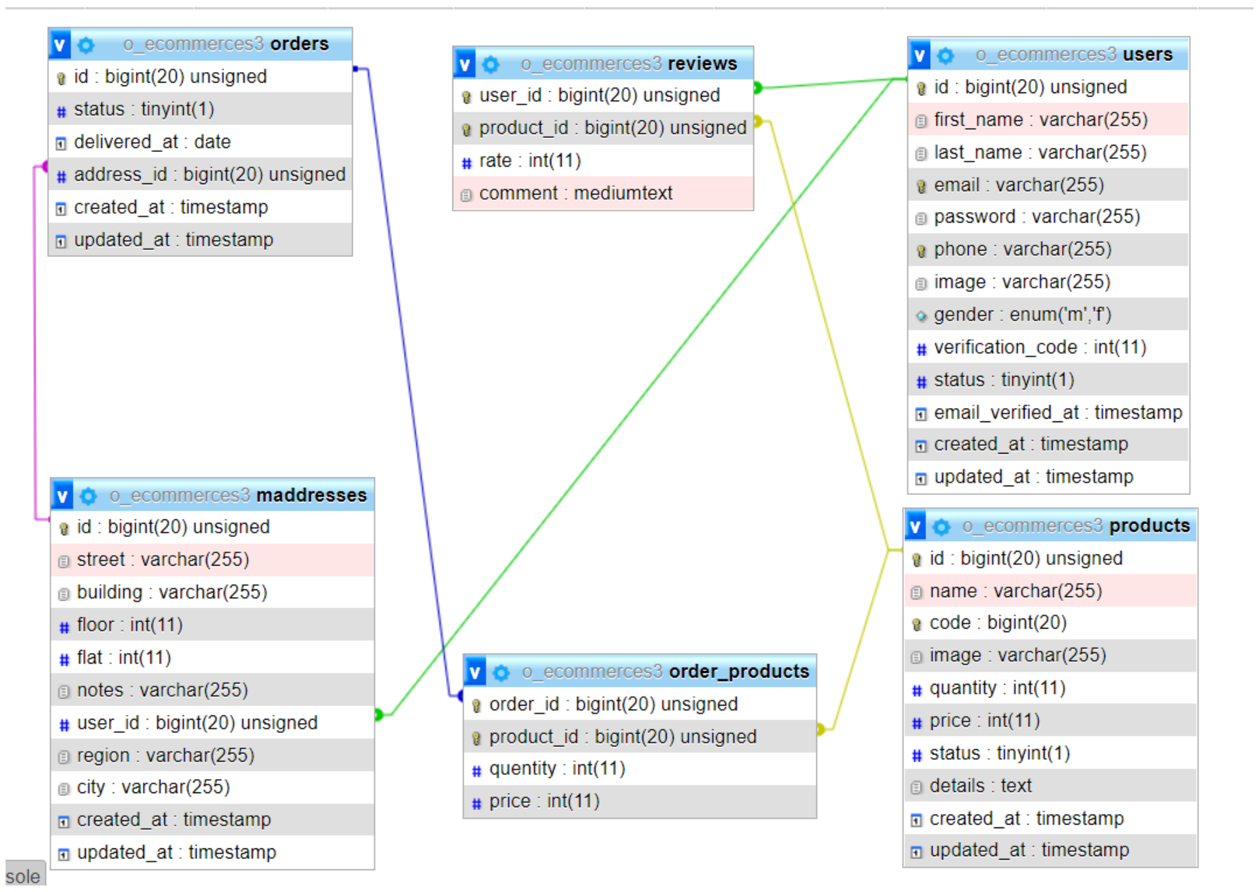


Database statistic



TABLE_NAME	TABLE_ROWS	AVG_ROW_LENGTH	DATA_LENGTH	MAX_DATA_LENGTH	INDEX_LENGTH	DATA_FREE	AUTO_INCREMENT
addresses	774663	75	58294272	0	55492608	6291456	1399177
cities	3050	53	163840	0	0	0	3231
orders	1040868	42	44662784	0	33062912	6291456	1879820
order_products	2096698	84	178110464	0	73023488	5242880	*NULL*
products	951375	119	113934336	0	23658496	4194304	1427979
regions	19980	79	1589248	0	1589248	4194304	22038
reviews	1251623	69	86671360	0	49938432	5242880	*NULL*
users	926236	146	135970816	0	72630272	5242880	2062200

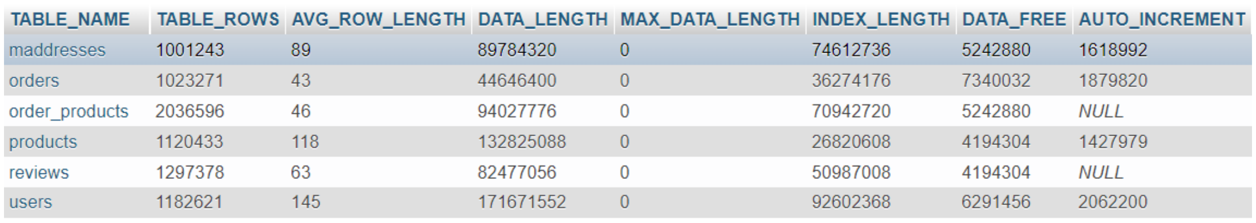
2. New schema



Join three tables cities, regions and addresses in one table call addresses that has column represent the region and another one represent the city for the address
This optimization change the database size from **139.5 MiB to 95.8 MiB decrease in memory requirement of approx 31%**

Also increase the efficiency of queries that need join between tables (cities,regions,addresses)

New Database statistic after modification



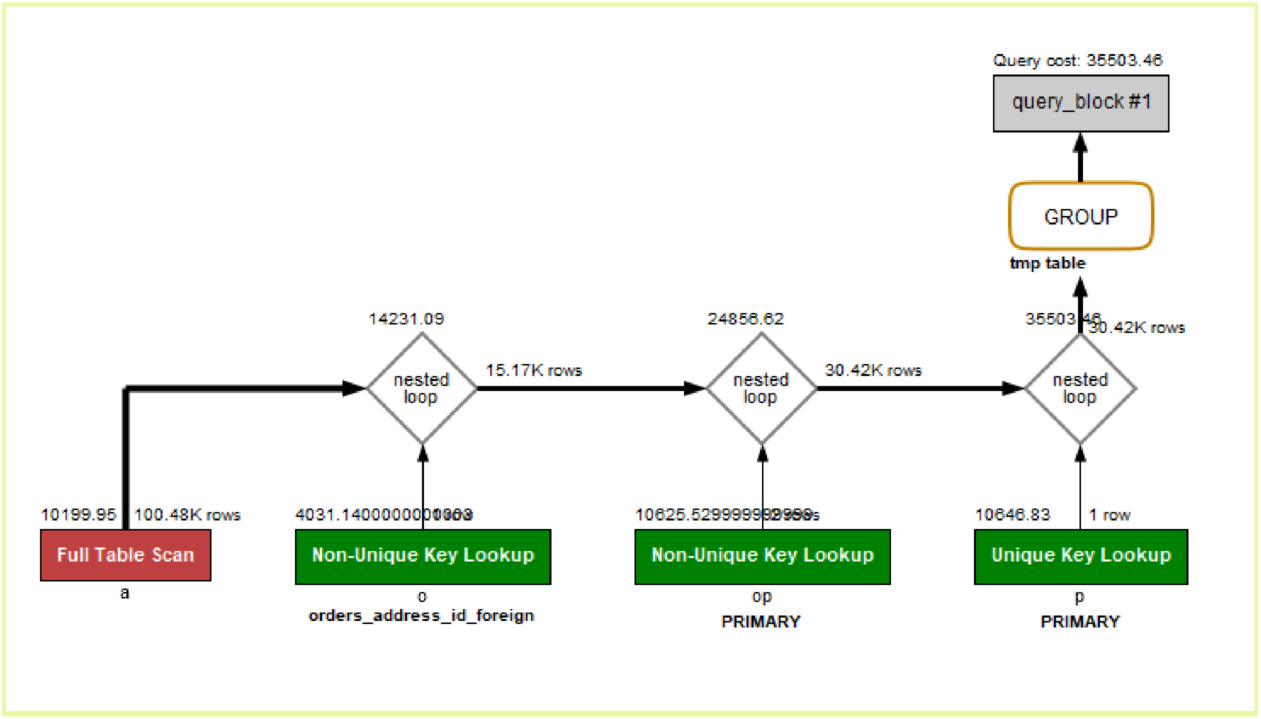
TABLE_NAME	TABLE_ROWS	AVG_ROW_LENGTH	DATA_LENGTH	MAX_DATA_LENGTH	INDEX_LENGTH	DATA_FREE	AUTO_INCREMENT
maddresses	1001243	89	89784320	0	74612736	5242880	1618992
orders	1023271	43	44646400	0	36274176	7340032	1879820
order_products	2036596	46	94027776	0	70942720	5242880	*NULL*
products	1120433	118	132825088	0	26820608	4194304	1427979
reviews	1297378	63	82477056	0	50987008	4194304	*NULL*
users	1182621	145	171671552	0	92602368	6291456	2062200

Queries for new schema

QUERY 1

```
select p.name,SUM(op.quentity) as order_times
from products p
join order_products op
  on op.product_id=p.id
join orders o
  on op.order_id=o.id
join addresses a
  on o.address_id=a.id
where a.city='city57'
GROUP BY p.id;
```

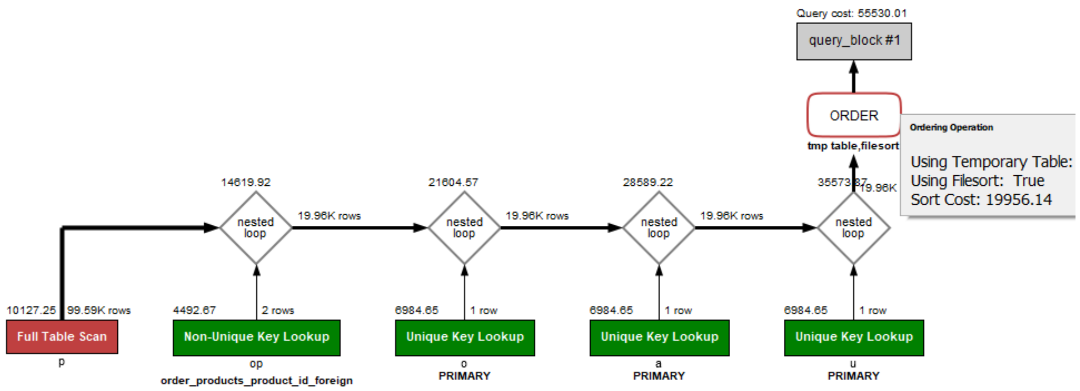
(Query took 1.3846 seconds.)



QUERY 2

Not changed

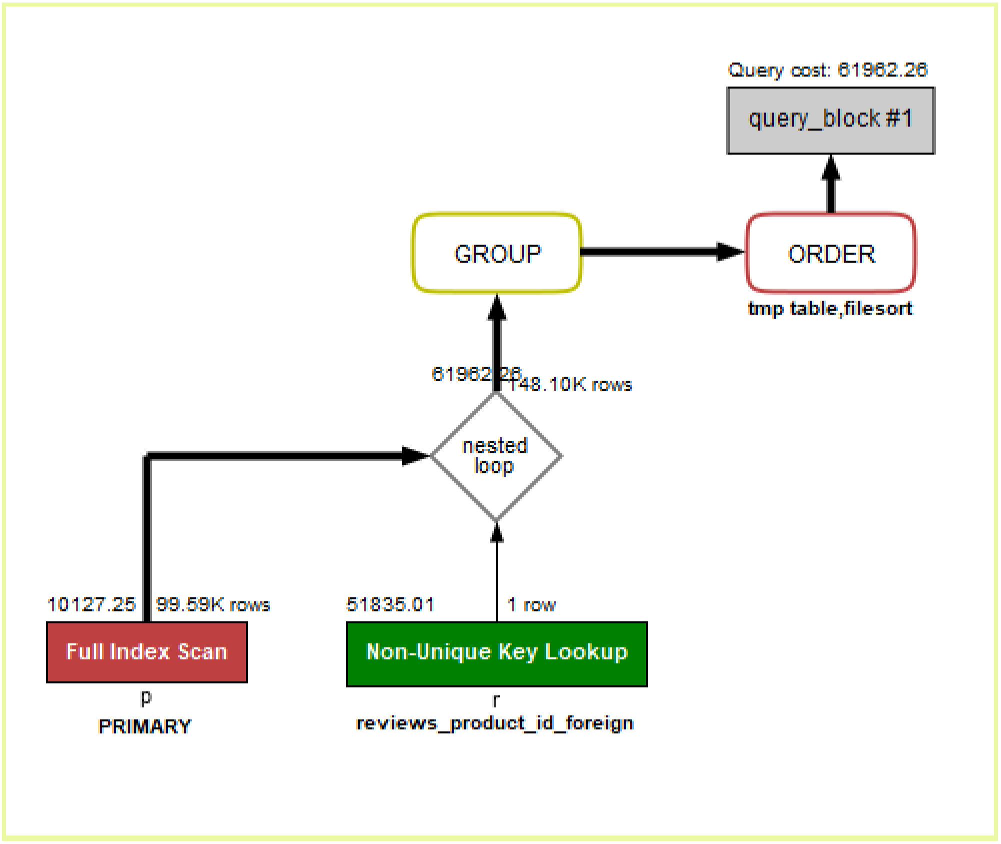
(Query took 0.2707 seconds)



QUERY 3

Not changed

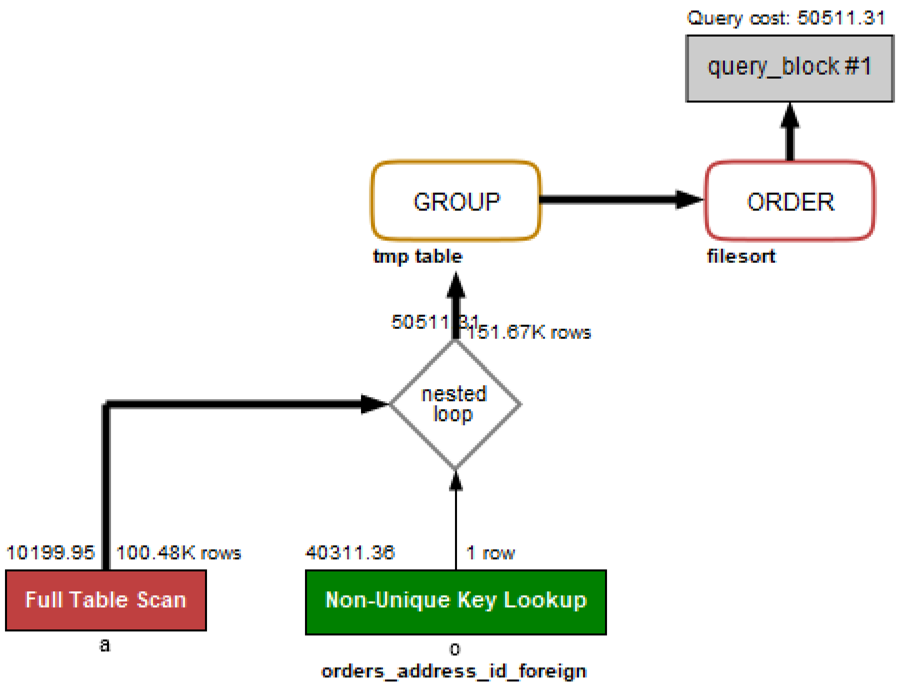
(Query took 2.1215 seconds.)



QUERY 4

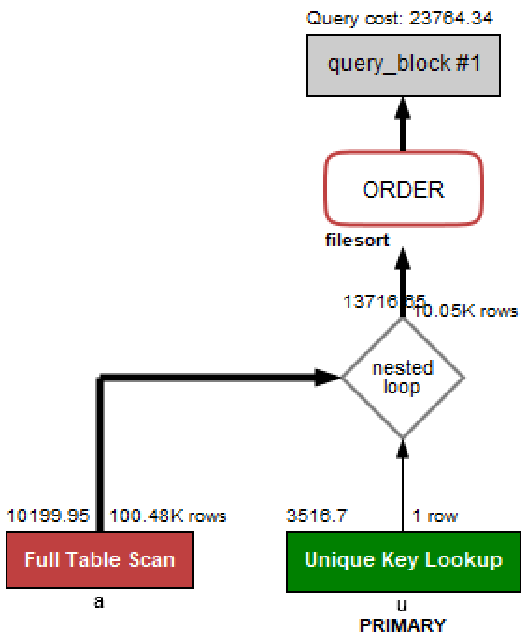
```
select a.city,Count(o.id)
as number_of_orders
from orders o
 join addresses a
on o.address_id=a.id
GROUP BY a.city
ORDER BY number_of_orders
DESC;
```

(Query took 1.8075 seconds.)



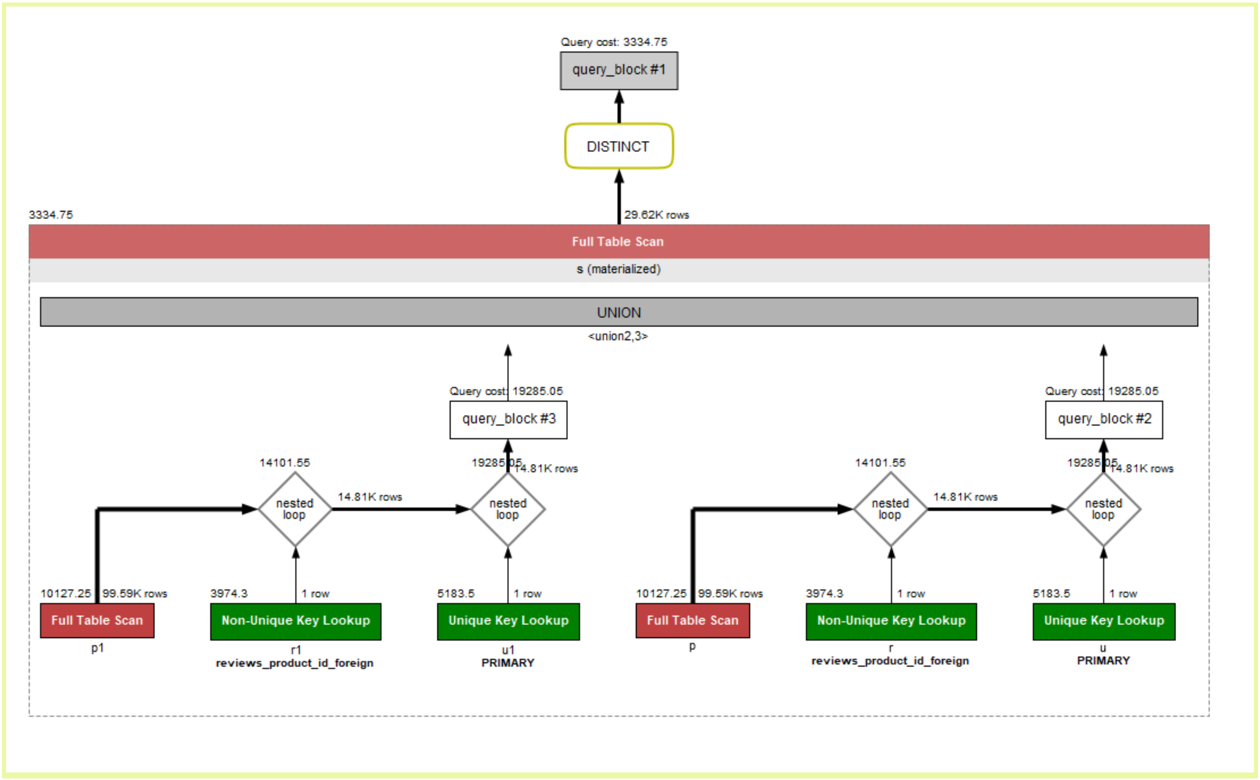
QUERY 5

```
select u.first_name,u.last_name,
a.street,a.building,a.floor
From addresses a
JOIN users u
 ON a.user_id=u.id
where a.region="region4660"
ORDER BY  a.street
```



QUERY 6

Not changed
(Query took 0.1654 seconds)



Effect of schema enhancement in 100K database (time enhancement)

	Time before schema optimization	Time after schema optimization	The percentage of time enhancement	comment
QUERY 1	Query took 3.5253 seconds.	Query took 1.3846 seconds.	60.7%	Compinning region and city in address table reduce the time of join between the city and the region and then between the region and the addresses
QUERY 2	Query took 0.556 seconds.	Query took 0.2707 seconds.	51%	It should be no difference as the change of the schema not affected this query
QUERY 3	Query took 2.1291 seconds	Query took 2.1215 seconds	.19%	It should be no difference as the change of the schema not affected this query
QUERY 4	(Query took 9.8346 seconds	Query took 1.8075 seconds	87.5%	Compinning region and city in address table reduce the time of join between the city and the region and then between the region and the addresses
QUERY 5	Query took 3.7409 seconds	Query took 0.2063 seconds.	90.1%	Compinning region in address table reduce the time of join between the city and the region and then between the region and the addresses
QUERY 6	Query took 0.1533 seconds.	Query took 0.1654 seconds	-7.8.%	It should be no difference as the change of the schema not affected this query

Effect of schema enhancement in 100K database(space enhancement)

	# rows processed before schema optimization	# rows processed after schema optimization	The percentage of space enhancement	comment
QUERY 1	2725	101160	-36.12	
QUERY 2	0	100009	-	
QUERY 3	349007	349007	0	
QUERY 4	288388	254228	0.118	
QUERY 5	23321	100007	-3.2	
QUERY 6	2	2	0	

Enhancement in memory management

SET GLOBAL innodb_buffer_pool_size=(2 * 1024 * 1024 * 1024);
Changed the buffer pool size
** The Buffer Pool caches pages that were recently accessed. If a lot of pages are being accessed sequentially, the Buffer Pool also preemptively caches nearby pages. Pages are evicted using a least recently used (LRU) algorithm.
Increase the buffer pool is increase number of pages cached in the buffer pool
Also we can change the innodb_page_size to increase the date in page than let be father to specific limit

	Time before memory optimization	Time after memory optimization	The percentage of time enhancement
QUERY 1	Query took 3.5253 seconds.	Query took 1.7862 seconds	49.33%
QUERY 2	Query took 0.556 seconds.	Query took 0.0047 seconds.	99.15%
QUERY 3	Query took 2.1291 seconds	Query took 0.7902 seconds	62.885%
QUERY 4	(Query took 9.8346 seconds	Query took 2.8662 seconds.)	70.855%
QUERY 5	Query took 3.7409 seconds	Query took 0.0127 seconds.	99.66%
QUERY 6	Query took 0.1533 seconds.	Query took 0.0139 seconds	90.932%

Modification in indexes

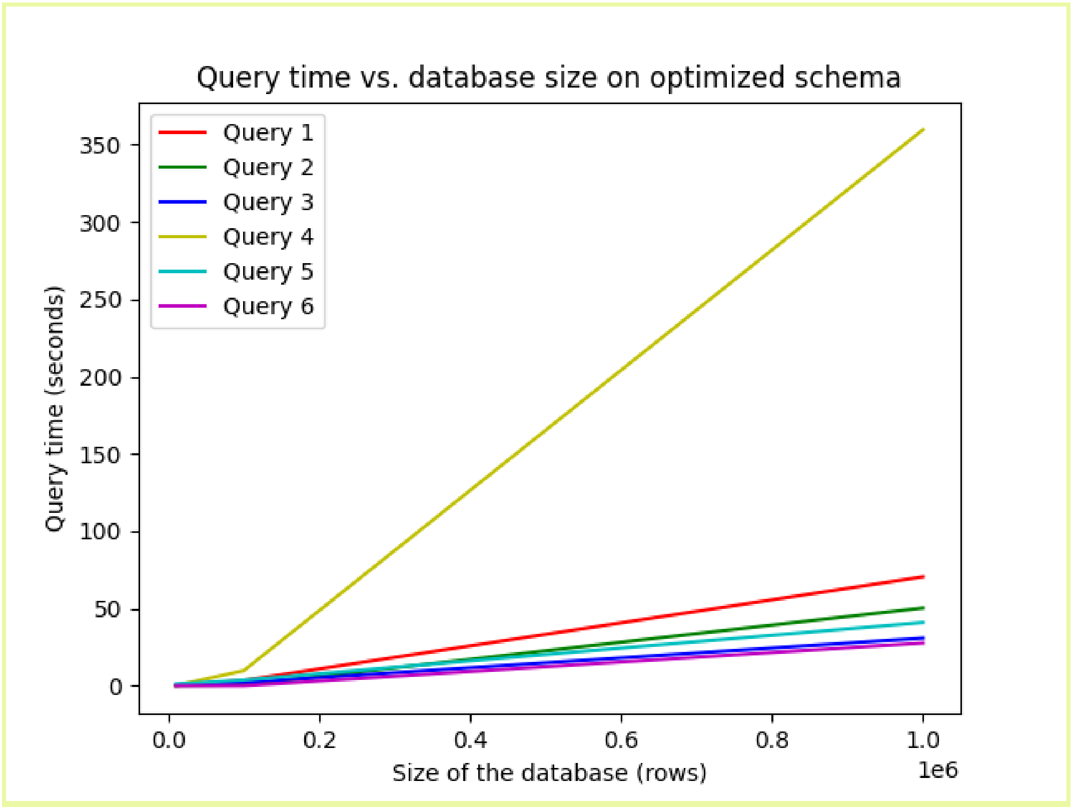
	Time before add index	Time after add index	The percentage of time enhancement	Added index & change in index size	comment
QUERY 1	Query took 3.5253 seconds.	3.5066 seconds	0.53%	Cities.name 752 KiB ->848 KiB	This index not affected the time too match as the size of cities table in small ~3.2k
QUERY 2	Query took 0.556 seconds.	Query took 0.0280 seconds After second index Query took 0.0054 seconds	94.964% After second index 99.028%	Orders.delivered_a And then add index to Products.name orders=> 24.6 MiB -> 28.1 MiB Products => 15 MiB -> 18.5 MiB	Index in the column in the where condition increase the performance Also index in the column we order in order increase the performance
QUERY 3	Query took 2.1291 seconds	_	_	All index need is already exist so to test the effect of the index in foreign key product_id Remove it and test the time to find it 5.5082 seconds	-
QUERY 4	Query took 9.8346 seconds	Query took 8.929 seconds	9.208%	Cities.name 752 KiB ->848 KiB	This index not affected the time too match as the size of cities table in small ~3.2k All index in foreign keys and primary keys is already exist
QUERY 5	Query took 3.7409 seconds	Query took 2.2312 seconds.	40.356%	regions.name 3.5 MIB -> 3 MIB addresses.street	Index in the column in the where condition increase the performance Also index in the column we order in order increase the performance

QUERY 6	Query took 0.1533 seconds.	Query took 0.1049 seconds	31.572%	Products.name 15 MiB -> 18.5 MiB	Index in the column in the where condition increase the performance

The effect of database size

1. **Schema non optimized**

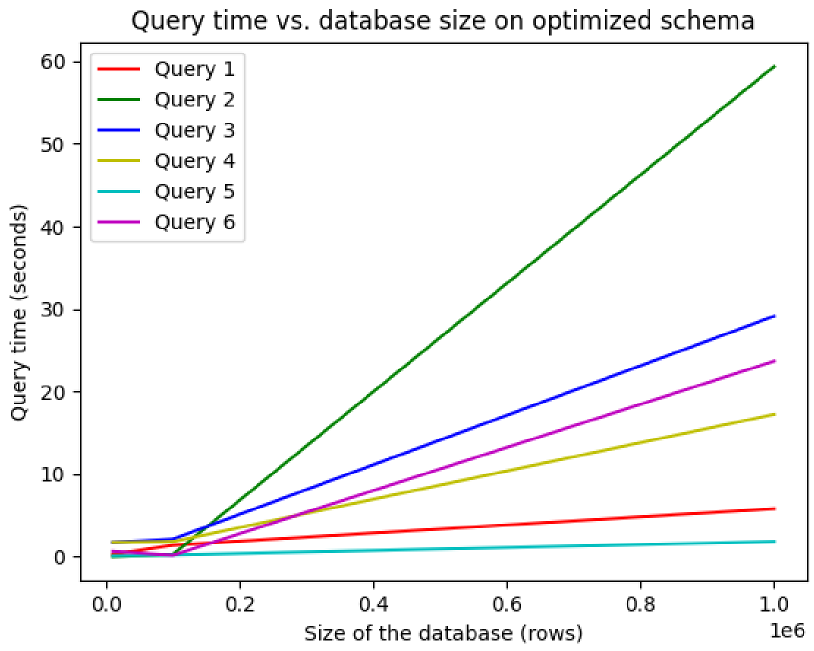
	10,000	100,000	1,000,000
QUERY 1	Query took 0.2541 seconds	Query took 3.5253 seconds.	Query took 70.5162 seconds
QUERY 2	Query took 0.2044 seconds.	Query took 0.556 seconds.	Query took 50.3569 seconds.*
QUERY 3	Query took 0.4223 seconds	Query took 2.1291 seconds	Query took 30.8541 seconds.
QUERY 4	Query took 0.2855 seconds	(Query took 9.8346 seconds	Query took 359.6192 seconds
QUERY 5	Query took 1.1439 seconds	Query took 3.7409 seconds	Query took 40.9703 seconds
QUERY 6	Query took 0.0432 seconds	Query took 0.073 seconds.	Query took 27.6998 seconds.



We see significant increase in time with increase the size of the database

2. **Schema optimized**

	10,000	100,000	1,000,000
QUERY 1	Query took 0.2861 seconds	Query took 1.3846 seconds.	Query took 5.7923 seconds
QUERY 2	Query took 0.0152 seconds	Query took 0.2707 seconds.	Query took 59.3569 seconds.*
QUERY 3	Query took 1.7250 seconds	Query took 2.1215 seconds	Query took 29.1147 seconds
QUERY 4	Query took 1.7330 seconds.	Query took 1.8075 seconds	Query took 17.2258 seconds
QUERY 5	Query took 0.1189 seconds	Query took 0.2063 seconds.	Query took 1.8139 seconds
QUERY 6	Query took 0.6459 seconds	Query took 0.1654 seconds	Query took 23.6700 seconds



We see significant increase in time with increase the size of the database but the increase in time is not high as the non optimized schema

NoSQL

Query 1

```
db.products.aggregate([
    //Join with user_info table
    {
        $lookup: {
            from: "order_products", // other table name
            localField: "id", // name of users table field
            foreignField: "product_id", // name of userinfo tablefield
            as: "product_order", // alias for userinfo table
        },
    },
    { $unwind: "$product_order" }, // $unwind used for getting data in
object or for one record only
    {
        $lookup: {
            from: "orders",
            localField: "product_order.order_id",
            foreignField: "id",
            as: "order",
        },
    },
    {$unwind: "$order"},
    {
        $lookup: {
            from: "addresses",
            localField: "order.address_id",
            foreignField: "id",
            as: "address",
        },
    },
    {$unwind: "$address"},
    {
        $match: {
            "address.city": "city1872",
        },
    },
    {
        $group: {
            _id: "id",
            order_times: { $sum: "$product_order.quentity" },
        },
    },
    {
        $project: {
            name: 1,
            order_times: 1,
        },
    },
]);
```

Query 2

```
db.products.aggregate([
    //Join with user_info table
    {
        $lookup: {
            from: "order_products", // other table name
            localField: "id", // name of users table field
            foreignField: "product_id", // name of userinfo tablefield
            as: "product_order", // alias for userinfo table
        },
    },
    { $unwind: "$product_order" },
    {
        $lookup: {
            from: "orders",
            localField: "product_order.order_id",
            foreignField: "id",
            as: "order",
        },
    },
    {$unwind: "$order"},
    {
        $lookup: {
            from: "addresses",
            localField: "order.address_id",
            foreignField: "id",
            as: "address",
        },
    },
    {$unwind: "$address"},
    {
        $lookup: {
            from: "users",
            localField: "address.user_id",
            foreignField: "id",
            as: "user",
        },
    },
    {$unwind: "$user"},
    {
        $match: {name: "plapla8" // product name},
    },
    {
        $project: {
            "user.first_name": 1, // 1 for showing and 0 for not showing
            "user.last_name": 1,
            "order.delovered_at": 1,
        },
    },
    {
        $sort: {
            "order.delovered_at": -1, // 1 for ascending order and -1 for
descending order
        },
    },
]);
```

Query 3

```
db.reviews.aggregate([
    //Join with user_info table
    {
        $lookup: {
            from: "products", // other table name
            localField: "product_id", // name of users table field
            foreignField: "id", // name of userinfo tablefield
            as: "product", // alias for userinfo table
        },
    },
    { $unwind: "$product" }, // $unwind used for getting data in object or
for one record only

    {
        $group: {
            _id: "$product.id",
            avg_rate: { $sum: "rate" },
        },
    },
    {
        $project: {
            "product.name": 1, // 1 for showing and 0 for not showing
            avg_rate: 1,
        },
    },
    {
        $sort: {
            avg_rate: -1, // 1 for ascending order and -1 for descending
order
        },
    },
]);
```

Query 4

```
db.orders.aggregate([
    //Join with user_info table
    {
        $lookup: {
            from: "addresses", // other table name
            localField: "address_id", // name of users table field
            foreignField: "id", // name of userinfo tablefield
            as: "address", // alias for userinfo table
        },
    },
    { $unwind: "$address" }, // $unwind used for getting data in object or
for one record only
    {
        $group: {
            _id: "$address.city",
            number_of_orders: { $count: { } },
        },
    },
    {
        $project: {
            number_of_orders: 1, // 1 for showing and 0 for not showing
            "city": 1,
        },
    },
    {
        $sort: {
            number_of_orders: -1, // 1 for ascending order and -1 for
descending order
        },
    },
]);
```

Query 5

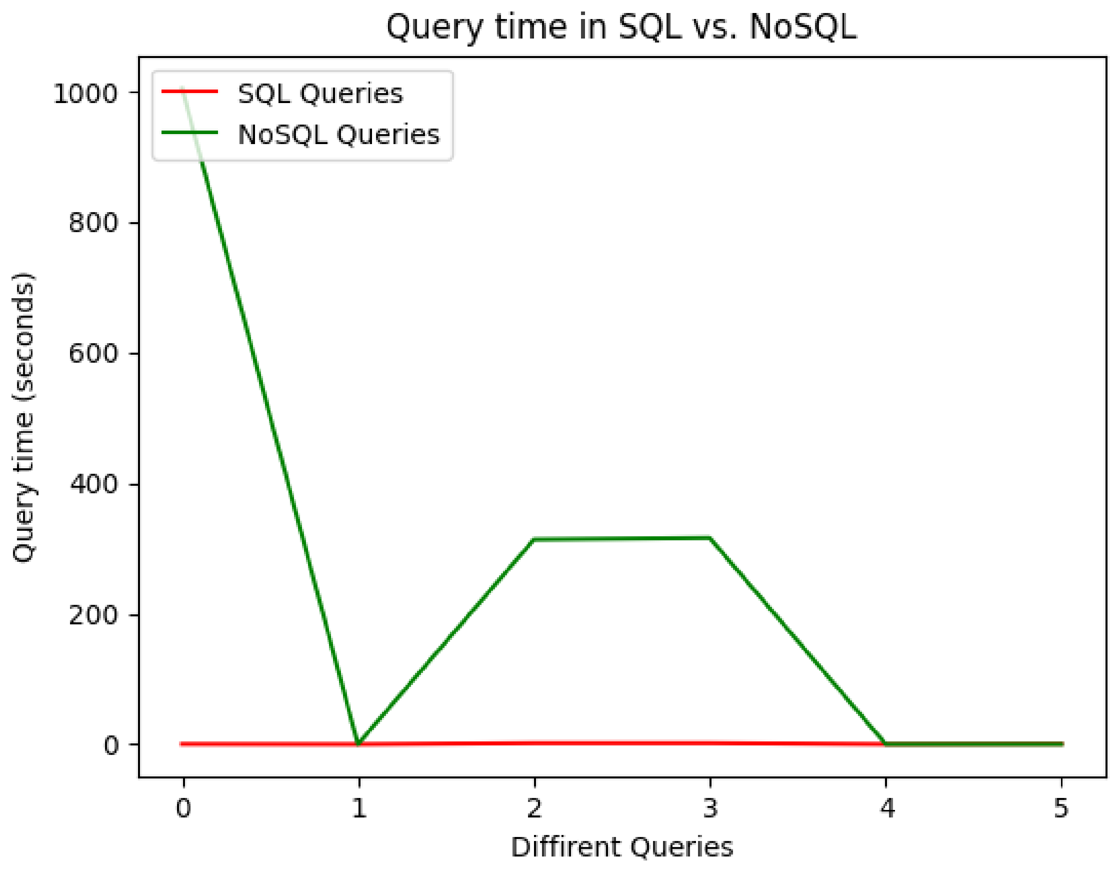
```
db.addresses.aggregate([
    //Join with user_info table
    {
        $lookup: {
            from: "users", // other table name
            localField: "user_id", // name of users table field
            foreignField: "id", // name of userinfo tablefield
            as: "users", // alias for userinfo table
        },
    },
    { $unwind: "$users" }, // $unwind used for getting data in object or
for one record only
    { $match : {"region" : "region5356"}},
    {
        $project: {
            "users.first_name": 1, // 1 for showing and 0 for not showing
            "users.last_name": 1,
            "street": 1,
            "building": 1,
            "floor": 1,
        },
    },
    {
        $sort: {
            street: -1, // 1 for ascending order and -1 for descending
order
        },
    },
]);
```

Query 6

```

db.products.aggregate([
    //Join with user_info table
    {
        $lookup: {
            from: "reviews", // other table name
            localField: "id", // name of users table field
            foreignField: "product_id", // name of userinfo tablefield
            pipeline : [
            {
                $lookup: {
                from: "users", // other table name
                localField: "user_id", // name of users table field
                foreignField: "id", // name of userinfo tablefield
                as: "users", // alias for userinfo table
                },

            },
            { $unwind: "$users" },
            ],
            as: "reviews", // alias for userinfo table
        },
    },
    { $unwind: "$reviews" }, // $unwind used for getting data in object or
for one record only

    {
         $match : { $or: [ {"name" : "plapla5"}, {"name" : "plapla2"} ]}
    },
    {
        $project: {
            "reviews.users.id": 1, // 1 for showing and 0 for not showing
            "reviews.users.first_name": 1,
            "reviews.users.last_name": 1
        },
    }
]);
```

SQL Vs NoSQL (volume 10000 row)

	SQL	NoSQL	comment
QUERY 1	Query took 0.2861 seconds	Query took 1005 seconds	Grouping in a NoSQL database is harder than SQL, because we use index in SQL. (index used in Mongo is a primary index to “_id” attribute)
QUERY 2	Query took 0.0152 seconds	Query took 0.14 seconds	In this query we needed to sort the retrieved rows which was easier using index, but it is still easy because the number of retrieved rows is small.
QUERY 3	Query took 1.7250 seconds	Query took 313.48 seconds	Agine grouping and sorting but with two joining tables.
QUERY 4	Query took 1.7330 seconds.	Query took 315.91 seconds	Same to Q3
QUERY 5	Query took 0.1189 seconds	Query took 0.04 seconds	Match specific region minimize the retrieved rows (highly selective attribute)
QUERY 6	Query took 0.8818 seconds	Query took 0.066 seconds	Same to Q5



SQL Vs NoSQL (volume 100000 row)

	SQL	NoSQL	comment
QUERY 1	Query took 1.3846 seconds.	-	Grouping in NoSQL database is harder than SQL, because of index in SQL
QUERY 2	Query took 0.2707 seconds.	Query took 52 seconds	
QUERY 3	Query took 2.1215 seconds	Query took > 1 hour	
QUERY 4	Query took 1.8075 seconds	Query took > 1 hour	
QUERY 5	Query took 0.2063 seconds.	Query took 0.037 seconds	

QUERY 6	Query took 0.1654 seconds	Query took 0.427 seconds	

Combine all enhancements together

1. 100k volume

	Query time before any optimization	After optimization	Percentage of time enhancement	Comment
Q1	Query took 3.9464 seconds.	Query took 1.9702 seconds.	50.1%	Add index to addresses.city
Q2	Query took 0.437 seconds	Query took 0.0313 seconds	93%	Add index to products.name
Q3	Query took 2.1291 seconds.	Query took 1.8245 seconds	14.3%	
Q4	Query took 9.8346 seconds	Query took 0.7198 seconds	93%	Add index to addresses.city
Q5	Query took 3.7409 seconds	Query took 0.0521 seconds	99%	Add index to addresses.regions and index to addresses.street
Q6	Query took 0.1533 seconds.	Query took 0.0189 seconds	88%	Add index to products.name

2. 1000k volume

	Query time before any optimization	After optimization	Percentage of time enhancement	Comment
Q1	Query took 70.5162 seconds	Query took 2.8969 seconds	95.89%	Add index to addresses.city
Q2	Query took 50.3569 seconds.*	Query took 0.969 seconds	98.07%	Add index to products.name

Q3	Query took 30.8541 seconds.	Query took 12.0721 seconds	60.873%	
Q4	Query took 359.6192 seconds	Query took 6.0818 seconds	98.30%	Add index to addresses.city
Q5	Query took 40.9703 seconds	Query took 0.8610 seconds	97.89%	Add index to addresses.regions and index to addresses.street
Q6	Query took 27.6998 seconds.	Query took 0.0004 seconds	99.99%	Add index to products.name

The results show that the combinations between different type of optimization show great results.
Query optimization has an effect when the DBMS didn't optimize the query and reorder the join to get min cost.

Index the table reduces the time for selection and grouping by on the volume which has the index, also when joining on the foreign key it reduces the time when there is an index.

Increasing the buffer pool increases the number of pages cached in the buffer pool which reduces the time of the query.

Also optimizing the schema by removing tables that cause overhead to be joined.