

Optimizacija upita u MySQL-u

Sistemi za upravljanje bazama podataka

MySQL Optimizator Upita

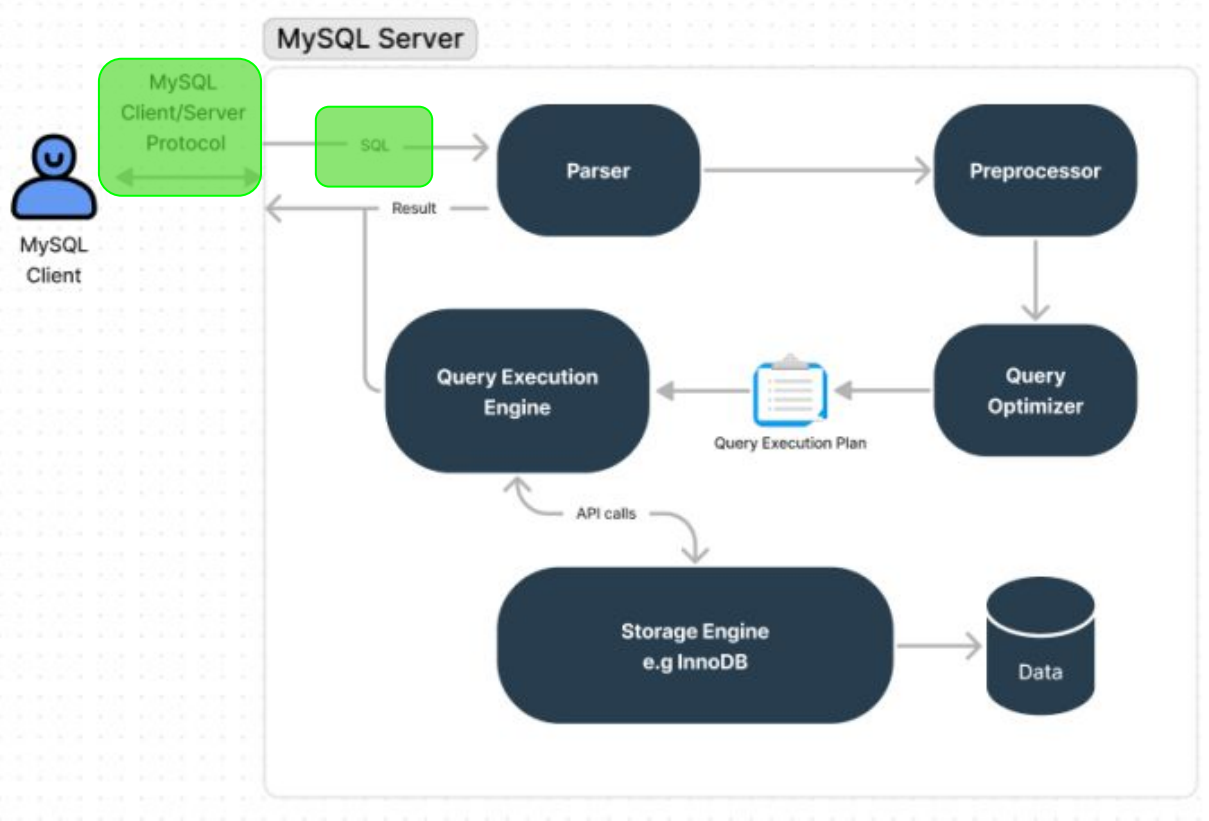
Softverski mehanizam za analizu SQL upita i izbor najefikasnijeg plana izvršenja upita.



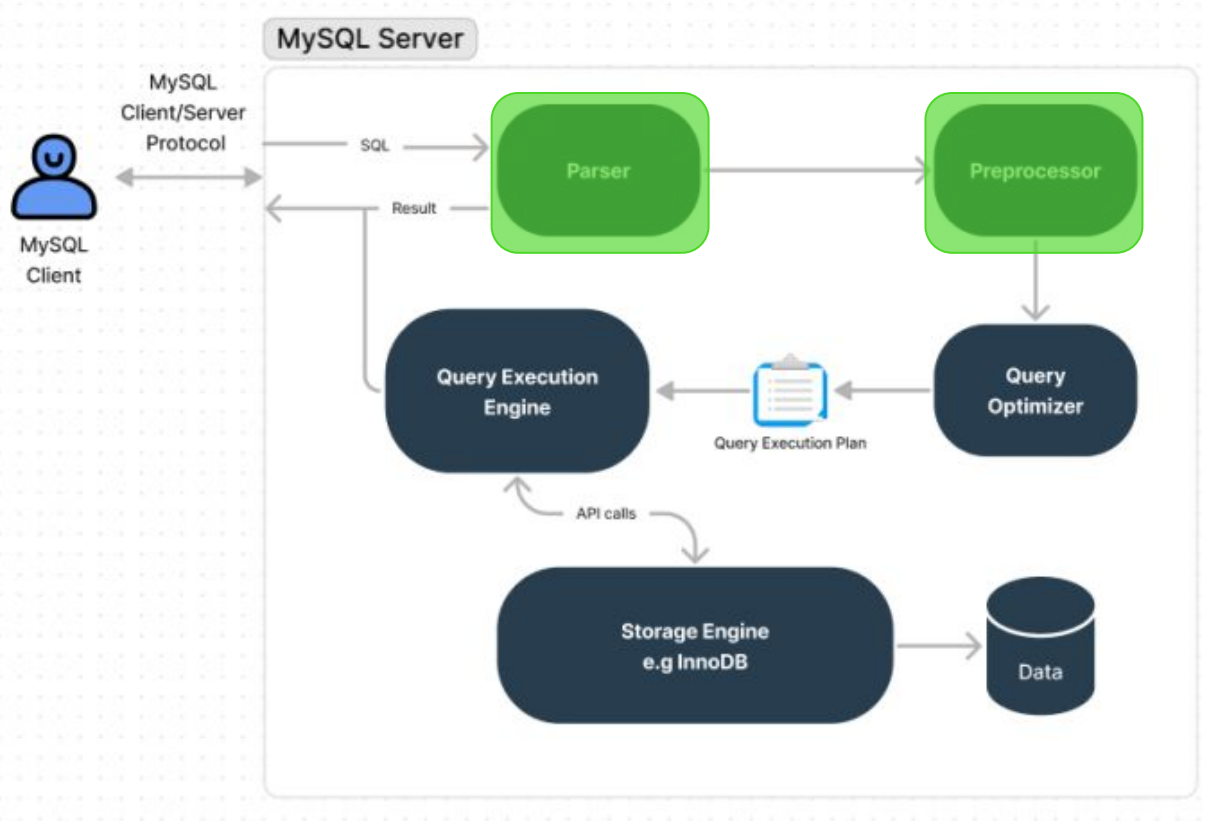
Šta je plan izvršenja?

Skup operacija koje optimizator bira za najefikasnije izvođenje upita. Plan izvršavanja je detaljan plan koji opisuje korake koje je potrebno preduzeti kako bi se izvršio upit. Plan izvršavanja obično uključuje određivanje kojim tabelama se treba pristupiti, kako se te tabele povezuju, koje uslove treba primeniti na podatke i koji se indeksi koriste.

Kratak pregled procesa izvršenja upita u MySQL-u



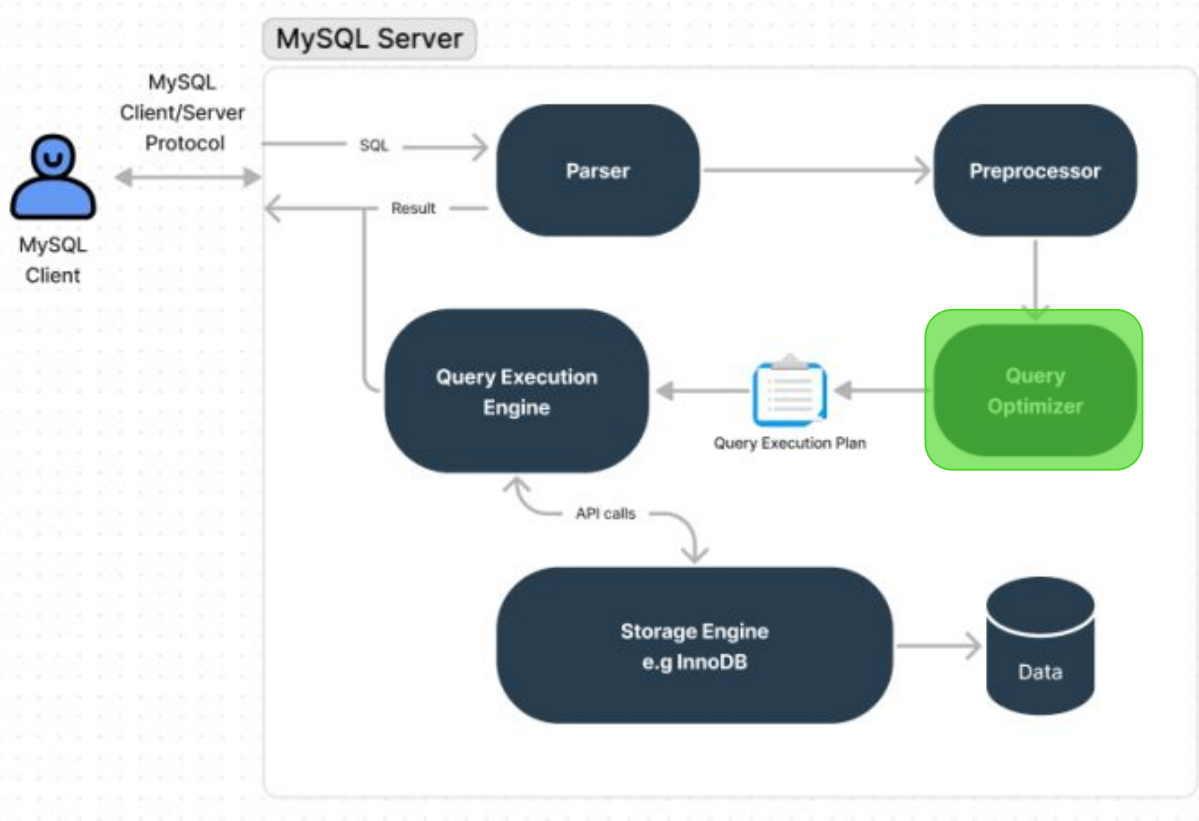
1.
MySQL klijent šalje upit do MySQL servera koristeći MySQL Client/Server protokol



2.

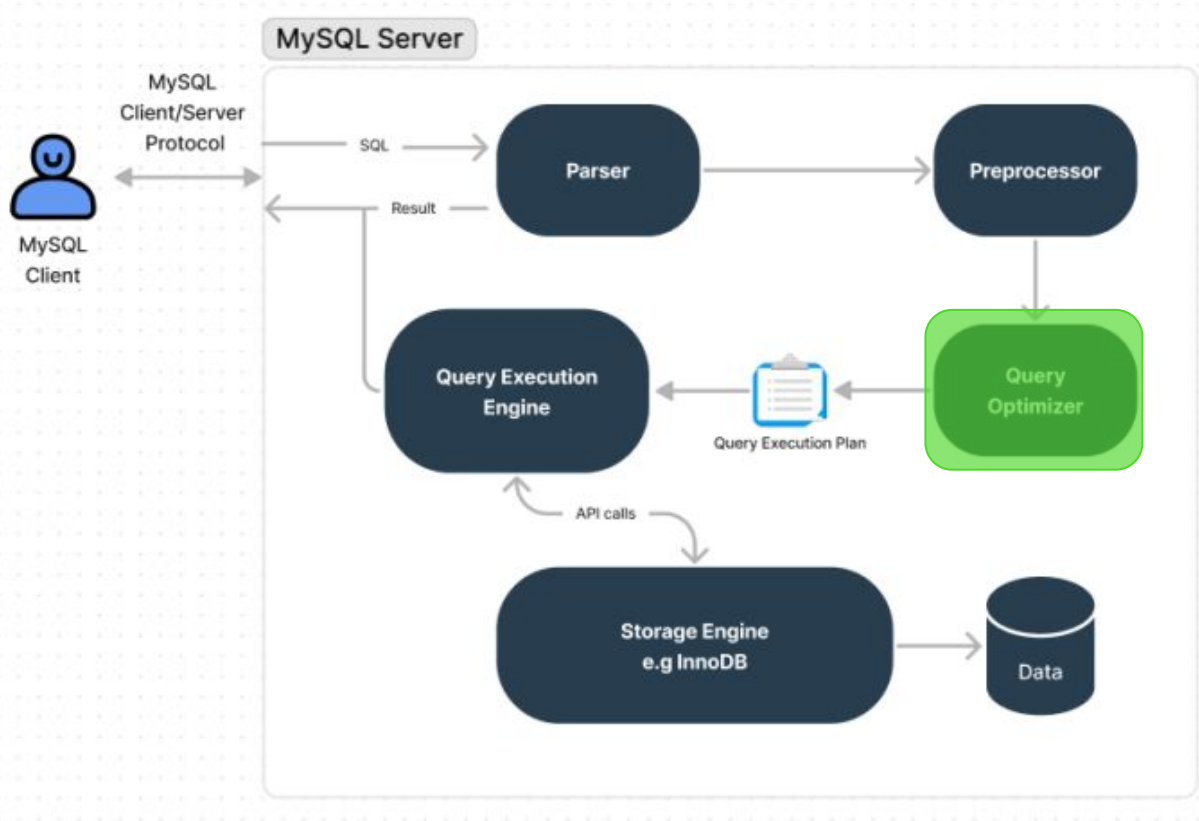
Parsiranje i preprocesiranje upita

Optimizator prvo parsira upit kako bi odredio njegovu sintaksu i semantiku, pa generiše internu reprezentaciju upita koju može koristiti za dalju obradu.



3. Analiza upita od strane optimizatora

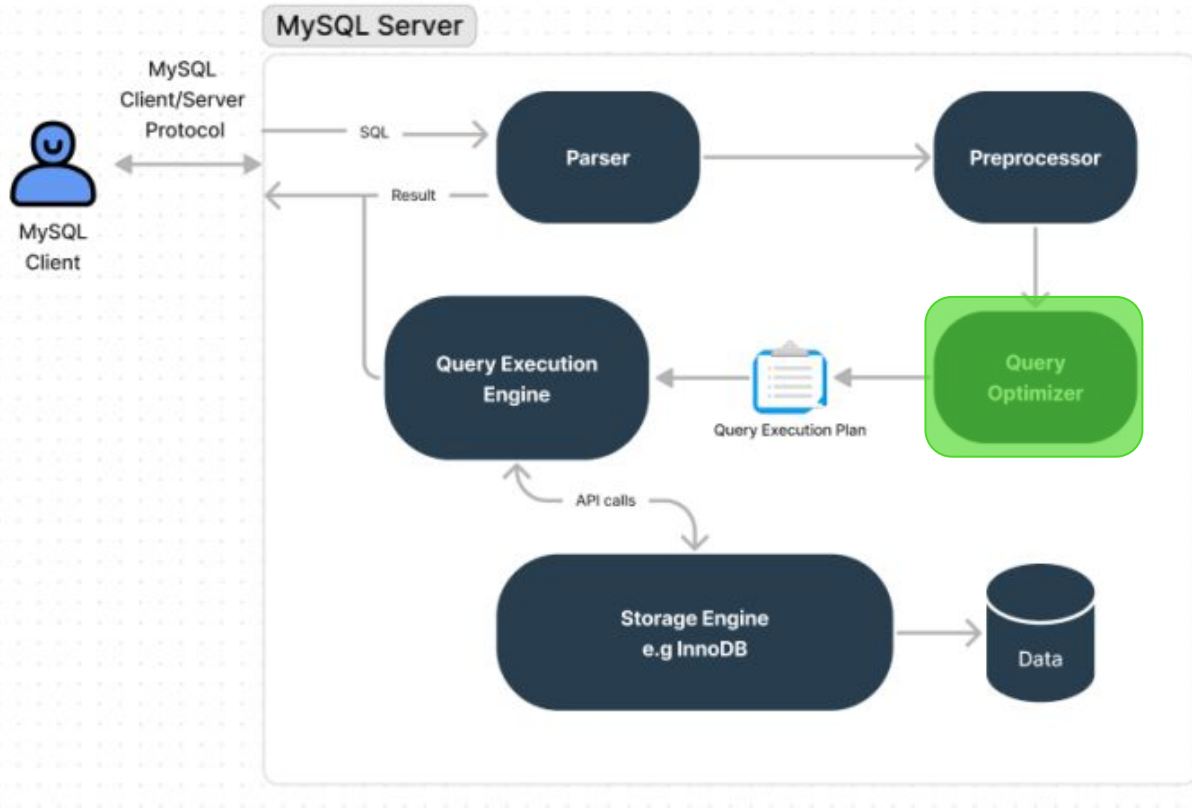
Optimizator vrši analizu strukture SQL upita tako što proverava da li je upit sintaksno i sematički korektan. Određuje koje tabele i kolone su uključene u upit, koji su dostupni indeksi i koje vrste su potrebne.



4.

Transformacija upita

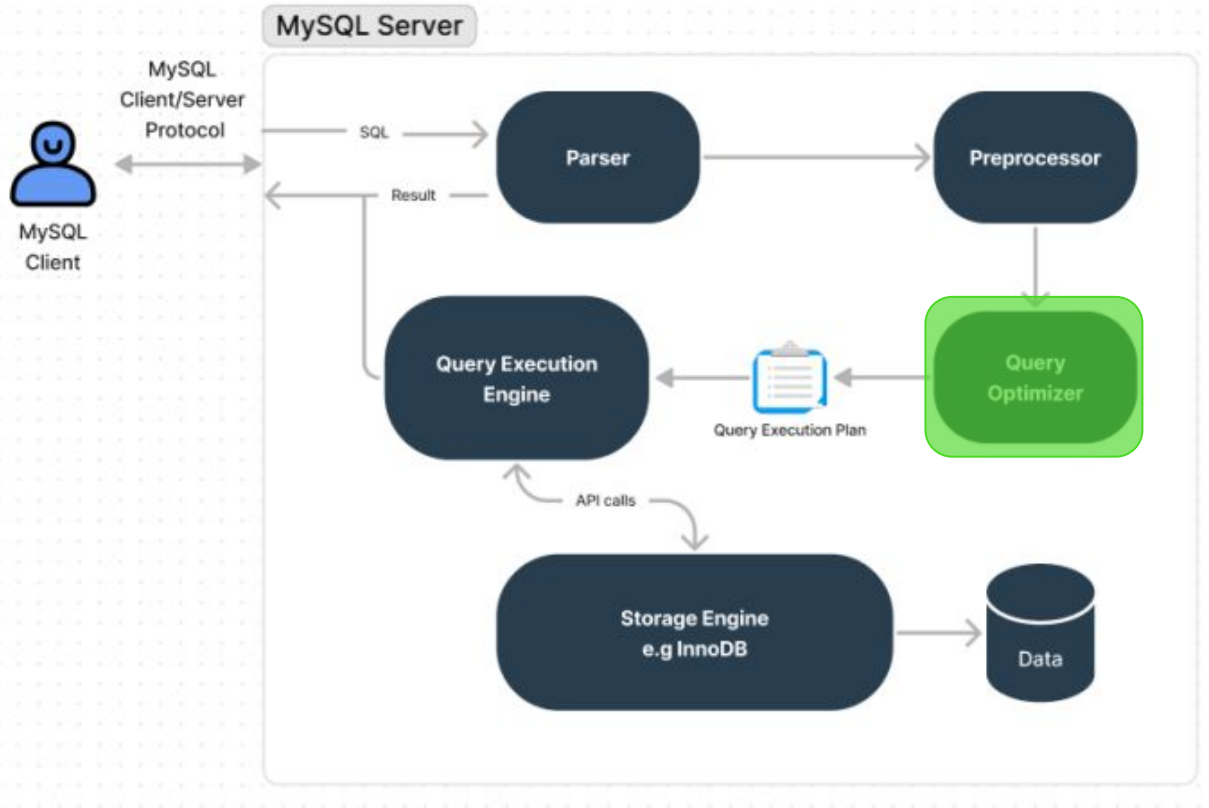
Optimizator transformiše upit u efikasniji oblik koji se može brže izvršavati



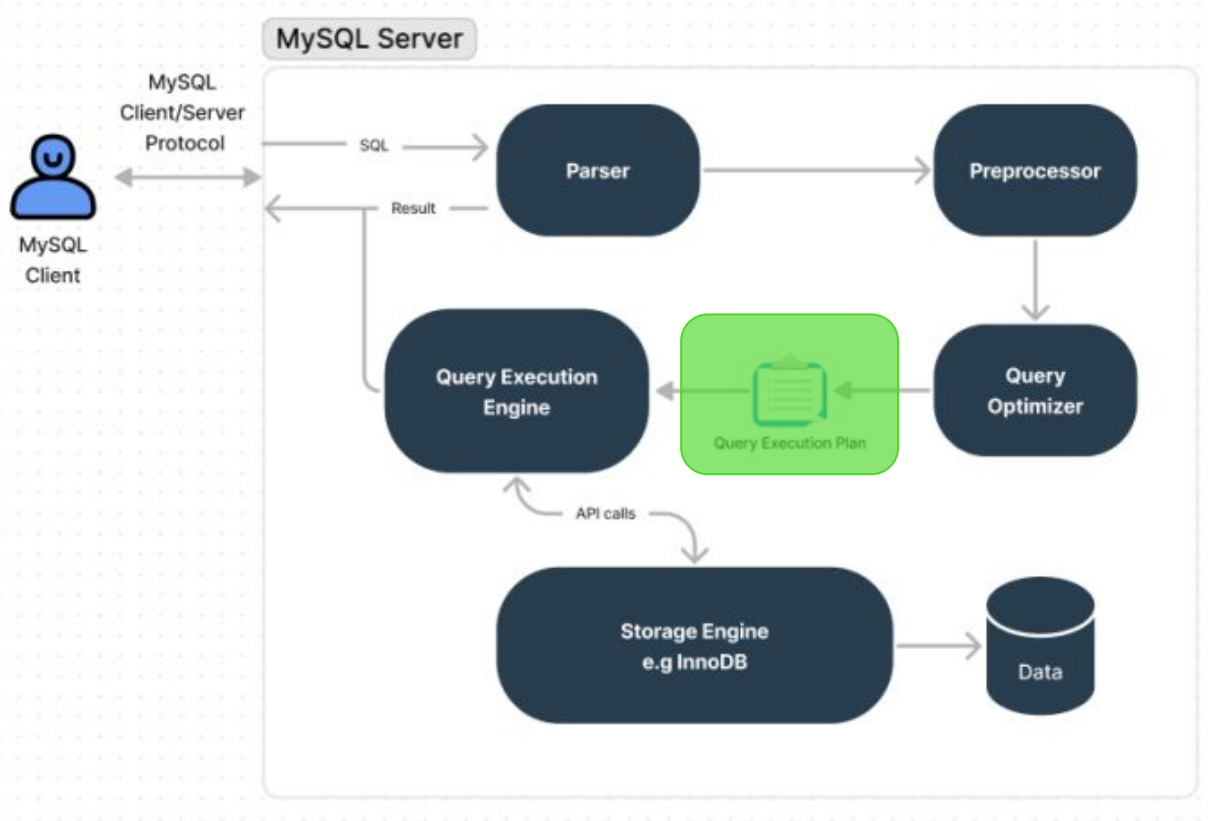
5.

Procena troškova izvršenja

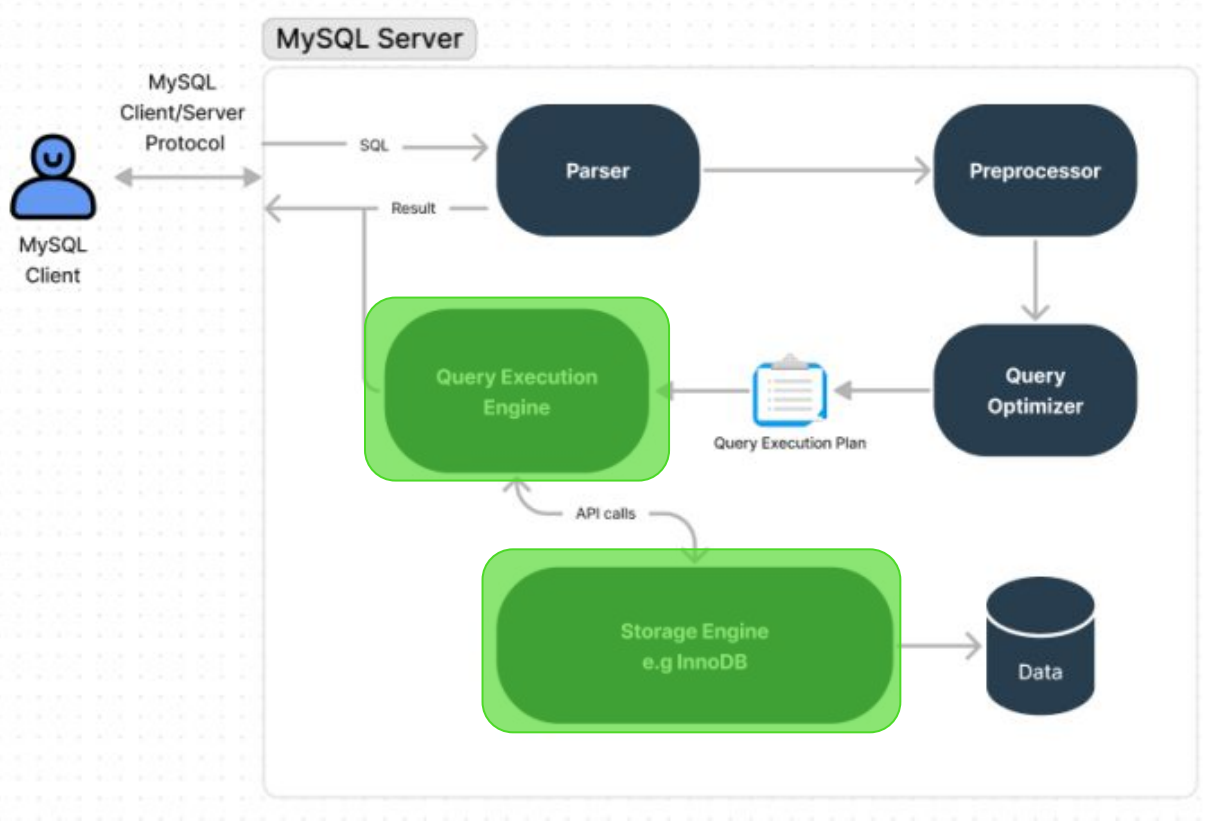
Optimizator vrši procenu troškova- I/O operacije, korišćenje procesora i memorije



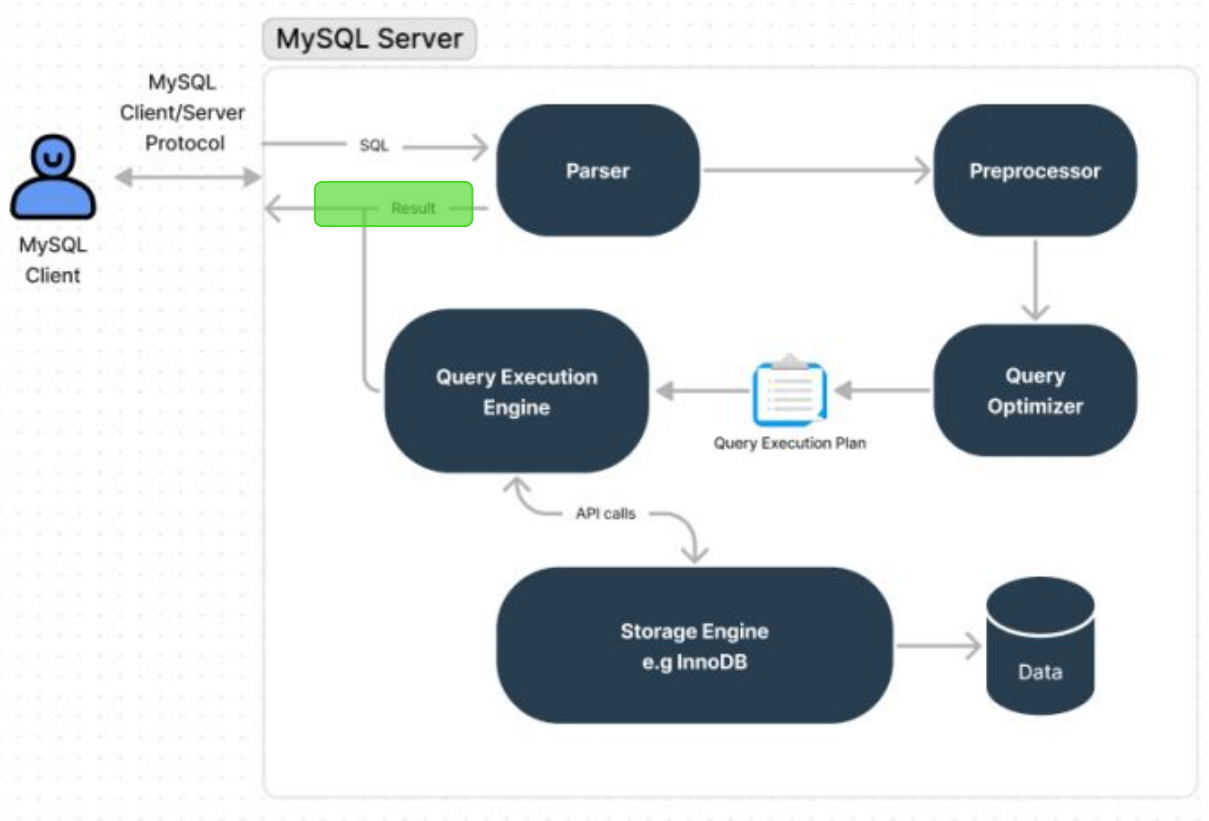
6.
Generisanje skupa mogućih
planova izvršenja, svaki sa
drugačijom procenom troškova



7.
Selekcija plana sa najnižom
procenom troškova



8.
Query Execution Engine izvršava
odabrani plan kreiranjem poiva ka
Storage Engine-u koji pristupa
podacima u bazi



9.
MySQL šalje rezultate ka MySQL
klijentu

Šta su indeksi?

Indeksi u MySQL DBMS-u su strukture koje se koriste za ubrzavanje pretrage podataka u bazi podataka. Indeksi se sastoje od jedne ili više kolona u tabeli baze podataka, a svaka kolona je sortirana po vrednosti.

Kada se pretražuje baza podataka, DBMS koristi indeks da bi brzo pronašao podatke koji odgovaraju zadatim kriterijumima pretrage. Bez indeksa, DBMS bi morao pregledati sve zapise u tabeli da bi pronašao odgovarajuće podatke, što bi bilo vrlo sporo za velike količine podataka.

MySQL Workbench

Local instance MySQL80

FileEditViewQueryDatabaseServerToolsScriptingHelp

Navigator

SCHEMAS

Filter objects

▼ sakila

Tables

actor

address

category

city

Columns

Indexes

PRIMARY

idx_fk_country_id

Foreign Keys

Triggers

country

customer

film

film_actor

film_category

film_text

inventory

language

payment

rental

staff

store

AdministrationSchemas

Information

Table: city

Columns:

city_id

city

country_id

last_update

smallint UN AI PK

varchar(50)

smallint UN

timestamp

Object InfoSession

Query 1SQL File 1SQL File 2

Limit to 1000 rows

1 • SET SESSION profiling = 1;

2 • use sakila;

3 • EXPLAIN ANALYZE SELECT first_name, last_name, city, country

4 FROM customer

5 INNER JOIN address USING(address_id)

6 INNER JOIN city USING(city_id) INNER JOIN country USING(country_id);

SQLAdditions

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Context HelpSnippets

Output

Action Output

#	Time	Action	Message	Duration / Fetch
---	------	--------	---------	------------------

Optimizacija upita **EXPLAIN** naredbom

EXPLAIN naredba u MySQL-u omogućava nam da vidimo kako će MySQL optimizator izvršiti određenu naredbu, što može biti veoma korisno u identifikovanju neefikasnosti u upitu

Plan izvršenja dobijen EXPLAIN naredbom daje ***mogući plan izvršenja*** za upit koji je naveden, a ne nužno tačan plan izvršenja.

```
EXPLAIN SELECT * FROM city WHERE city = 'Nis';
```

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
►	1	SIMPLE	city	HULL	ALL	HULL	HULL	HULL	HULL	600	10.00	Using where

Optimizacija upita **EXPLAIN** naredbom

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	city	NULL	ALL	NULL	NULL	NULL	NULL	600	10.00	Using where

Redni broj koji označava redosled u kome će MySQL izvršiti operacije u upitu

Optimizacija upita EXPLAIN naredbom <<<<<<

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	city	NULL	ALL	NULL	NULL	NULL	NULL	600	10.00	Using where

Označava vrstu SELECT naredbe koja se izvršava, može biti:

- o SIMPLE – obična SELECT naredba bez unija i podupita
- o PRIMARY – spoljni upit
- o UNION – druga ili kasnija SELECT naredba u uniji
- o SUBQUERY – prva SELECT naredba u podupitu

Optimizacija upita EXPLAIN naredbom

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	city	NULL	ALL	NULL	NULL	NULL	NULL	600	10.00	Using where

Označava ime tabele na koju se odnosi jedna vrsta u izlazu, a može imati i sledeće vrednosti

Optimizacija upita EXPLAIN naredbom

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	city	NULL	ALL	NULL	NULL	NULL	NULL	600	10.00	Using where

- Pokazuje broj particija (delova) tabele koje se koriste prilikom izvršavanja određenog upita.
- Particije su delovi tabele koje se mogu koristiti za organizaciju podataka po nekom kriterijumu, na primer po vremenu, geografskoj lokaciji ili vrednosti ključa.
- Kada se koriste particije, MySQL može brže pretraživati velike tabele, jer umesto pretraživanja cele tabele, pretražuje samo određeni deo. To može dovesti do značajnog poboljšanja performansi u određenim slučajevima.
- Ako se prikazuje "NULL", to znači da se particije ne koriste za ovaj upit. Ukoliko se prikazuje broj veći od 1, to znači da se koristi više particija

Optimizacija upita EXPLAIN naredbom

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	city	NULL	ALL	NULL	NULL	NULL	NULL	600	10.00	Using where

označava tip spoja u izrazu (join type) i određuje na koji način se pristupa podacima u bazi. Odnosi se na sve uslovne izraze, ne samo na spojeve između tabela koje će MySQL koristiti za pribavljanje podataka. Moguće vrednosti su:

- system
- const
- eq_ref
- ref
- fulltext
- ref_or_null
- index_merge
- unique_subquery
- Index_subquery
- range
- index
- ALL

Optimizacija upita EXPLAIN naredbom

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
►	1	SIMPLE	city	NULL	ALL	NULL	NULL	NULL	NULL	600	10.00	Using where

Kaže nam koji su indeksi mogući za pribavljanje podataka iz tabele. Ova kolona je potpuno nezavisna od redosleda tabela koji se prikazuju u izlazu iz EXPLAIN-a. Neki ključevi u possible_keys mogu da budu neupotrebljivi u praksi.

Optimizacija upita EXPLAIN naredbom <<<<<

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	city	NULL	ALL	NULL	NULL	NULL	NULL	600	10.00	Using where

Označava koji je indeks zapravo MySQL odlučio da koristi za pribavljanje podataka iz tabele. Ako MySQL odluči da koristi jedan od indeksa iz possible_keys za pretraživanje redova, taj će indeks biti naveden kao vrednost ključa ili key-a. Može se desiti da key bude indeks koji nije prisutan u possible_keys, ako nijedan od mogućih ključeva nije prikladan za pretraživanje redova, ali su sve kolone izabrane upitom kolone nekog drugog indeksa. U većini slučajeva, pretraživanje indeksa je efikasnije od pretraživanja redova podataka.

Optimizacija upita EXPLAIN naredbom

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
►	1	SIMPLE	city	NULL	ALL	NULL	NULL	NULL	NULL	600	10.00	Using where

Dužina indeksa tj. ključa koji će se koristiti za pribavljanje podataka iz tabele. Ako je kolona key jednaka NULL, onda je i key_len NULL

Optimizacija upita **EXPLAIN** naredbom

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	city	NULL	ALL	NULL	NULL	NULL	NULL	600	10.00	Using where

Označava kolonu ili konstantu koja će se koristiti sa key indeksom da bi se pribavili podaci iz tabele. Ako je vrednost func, vrednost koja se koristi je rezultat neke funkcije ili operatora

Optimizacija upita **EXPLAIN** naredbom

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	city	NULL	ALL	NULL	NULL	NULL	NULL	600	10.00	Using where

Procenjeni broj redova koje će MySQL morati procesuirati kako bi dobio konačan rezultat upita

Optimizacija upita EXPLAIN naredbom

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	city	NULL	ALL	NULL	NULL	NULL	NULL	600	10.00	Using where

Prikazuje procenat redova tabele koji su filtrirani uslovom tabele. Maksimalna vrednost je 100, što znači da nije bilo filtriranja redova. Vrednosti koje se smanjuju od 100 ukazuju na povećanje količine filtriranja.

Kolona rows prikazuje procenjeni broj redova koji su pregledani, a rows x filtered prikazuje broj redova koji su povezani sa sledećom tabelom.

Primer: ako je rows = 1000, filtered = 50%, broj redova koji će biti povezani sa sledećom tabelom je $1000 \times 50\% = 500$ redova.

Optimizacija upita EXPLAIN naredbom

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
►	1	SIMPLE	city	NULL	ALL	NULL	NULL	NULL	NULL	600	10.00	Using where

Dodatne informacije o tome kako će MySQL izvršiti upit, kao što je na primer sortiranje

Optimizacija upita OPTIMIZER_TRACE tabelom



Tabela OPTIMIZER_TRACE u MySQL-u se koristi za prikazivanje detalja o tome kako MySQL query optimizer izvršava upit. Dok EXPLAIN naredba daje prikaz odabranog plana izvršenja, OPTIMIZER_TRACE daje objašnjenje zašto je taj plan izabran.

```
1 • SET optimizer_trace='enabled=on';  
2 • select * from actor where first_name = 'JOHN';  
3 • select * from information_schema.optimizer_trace;
```

```
Binary Text JSON
1 {
2   "steps": [
3     {
4       "join_preparation": {
5         "select#": 1,
6         "steps": [
7           {
8             "expanded_query": "/* select#1 */ select `actor`.`actor_id` AS `actor_id`,`actor`.`first_name` AS `first_name`,`actor`.`last_name` AS `last_name`,`actor`.`last_update` AS `last_update` from `acto"
9           }
10        ]
11      }
12    },
13    {
14      "join_optimization": {
15        "select#": 1,
16        "steps": [
17          {
18            "condition_processing": {
19              "condition": "WHERE",
20              "original_condition": "(`actor`.`first_name` = 'JOHN')",
21              "steps": [
22                {
23                  "transformation": "equality_propagation",
24                  "resulting_condition": "multiple equal('JOHN', `actor`.`first_name`)"
25                },
26                {
27                  "transformation": "constant_propagation",
28                  "resulting_condition": "multiple equal('JOHN', `actor`.`first_name`)"
29                },
30                {
31                  "transformation": "trivial_condition_removal",
32                  "resulting_condition": "multiple equal('JOHN', `actor`.`first_name`)"
33                }
22
```

Optimizacija upita EXPLAIN ANALYZE naredbom



Ova naredba je detaljnija i daje statistike koje ne daje obična EXPLAIN naredba, a za te statistike je neophodno da se zapravo izvrši upit. Primer poziva komande i rezultat koji se dobija je:

```
use sakila;  
EXPLAIN ANALYZE SELECT first_name, last_name, city, country  
FROM customer  
INNER JOIN address USING(address_id)  
INNER JOIN city USING(city_id) INNER JOIN country USING(country_id);
```

```
-> Nested loop inner join (cost=643 rows=604) (actual time=0.469..6.06 rows=599 loops=1)  
  -> Nested loop inner join (cost=432 rows=604) (actual time=0.271..3.22 rows=603 loops=1)  
    -> Nested loop inner join (cost=221 rows=600) (actual time=0.185..1.79 rows=600 loops=1)  
      -> Table scan on country (cost=11.2 rows=109) (actual time=0.0741..0.127 rows=109 loops=1)  
        -> Index lookup on city using idx_fk_country_id (country_id=country.country_id) (cost=1.38 rows=5.5) (actual time=0.00702..0.0149 rows=5.5 loops=109)  
          -> Covering index lookup on address using idx_fk_city_id (city_id=city.city_id) (cost=0.25 rows=1.01) (actual time=0.00187..0.00229 rows=1 loops=600)  
            -> Index lookup on customer using idx_fk_address_id (address_id=address.address_id) (cost=0.25 rows=1) (actual time=0.00415..0.00456 rows=0.993 loops=603)
```


Optimizacija SELECT ... WHERE klauzule

Koraci koje preduzima MySQL optimizator

Uklanjanje nepotrebnih zagrada ili suvišnog koda



Pri eliminaciji suvišnog koda optimizator detektuje upite sledećeg tipa:

```
SELECT...WHERE 1=1 AND col1='value' AND 'apple'='(' apple'
```

I njih transformiše u sledeći oblik:

```
SELECT...WHERE col1='value'
```

Primer uklanjanja nepotrebnih zagrada:

Originalni upit: $((a \text{ AND } b) \text{ AND } c \text{ OR } (((a \text{ AND } b) \text{ AND } (c \text{ AND } d))))$

Transformisan upit: $(a \text{ AND } b \text{ AND } c) \text{ OR } (a \text{ AND } b \text{ AND } c \text{ AND } d)$

```
1 • SET optimizer_trace='enabled=on';
2 • select * from actor where 1 = 1 AND first_name = 'JOHN';
3 • select * from information_schema.optimizer_trace;
```

```
"steps": [
  {
    "condition_processing": {
      "condition": "WHERE",
      "original_condition": "(`actor`.`first_name` = 'JOHN')",
      "steps": [
        {
          "transformation": "equality_propagation",
          "resulting_condition": "(multiple equal('JOHN', `actor`.`first_name`))"
        },
        {
          "transformation": "constant_propagation",
          "resulting_condition": "(multiple equal('JOHN', `actor`.`first_name`))"
        },
        {
          "transformation": "trivial_condition_removal",
          "resulting_condition": "multiple equal('JOHN', `actor`.`first_name`)"
        }
      ]
    }
  ]
}
```

Uklanjanje preklopljenih konstanti

MySQL vrši i automatsko uklanjanje preklopljenih konstanti, primeri:

Originalni upit: `(a<b AND b=c) AND a=5`

Transformisan: `b>5 AND b=c AND a=5`

Originalni upit: `SELECT ... WHERE col1 = 2 * 5`

Transformisan: `SELECT ... WHERE col1 = 10`

Uklanjanje konstantnih uslova <<<<<

MySQL vrši i automatsko uklanjanje konstantnih uslova, primeri:

Originalni upit: (b>=5 AND b=5) OR (b=6 AND 5=5) OR (b=7 AND 5=6)

Transformisani: b=5 OR b=6

Pri eliminaciji konstantnih uslova, ono što treba imati na umu jeste zakon tranzitivnosti. Ukoliko je dat upit koji je oblika:

```
SELECT ... WHERE col1 <operator> col2 AND col2 <operator> 'value'
```

On se transformiše u upit sledećeg oblika:

```
SELECT ... WHERE col1 <operator> 'value' AND col2 <operator> 'value'
```

Pri čemu je operator $\in \{ =, >, <, \geq, \leq, <>, <=>, \text{LIKE} \}$.

- 1 • SET optimizer_trace='enabled=on';
- 2 • select * from city where country_id = city_id AND city_id = 1;
- 3 • select * from information_schema.optimizer_trace;

```

"steps": [
  {
    "condition_processing": {
      "condition": "WHERE",
      "original_condition": "((`city`.`country_id` = `city`.`city_id`) and (`city`.`city_id` = 1))",
      "steps": [
        {
          "transformation": "equality_propagation",
          "resulting_condition": "(multiple equal(1, `city`.`country_id`, `city`.`city_id`))"
        },
        {
          "transformation": "constant_propagation",
          "resulting_condition": "(multiple equal(1, `city`.`country_id`, `city`.`city_id`))"
        },
        {
          "transformation": "trivial_condition_removal",
          "resulting_condition": "multiple equal(1, `city`.`country_id`, `city`.`city_id`)"
        }
      ]
    }
  }
]

```

Optimizacija ORDER BY klauzule

Koraci koje preduzima MySQL optimizator

MySQL vrši i automatsko uklanjanje besmislenih ORDER BY klauzula kao što je sledeća:

```
1 • use sakila;  
2 • SET optimizer_trace='enabled=on';  
3 • EXPLAIN SELECT description FROM film ORDER BY 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	film	NULL	ALL	NULL	NULL	NULL	NULL	1000	100.00	Using filesort

Ključna stvar kod ORDER BY klauzule je da optimizator ume da prepozna uređenost vrsta i ukoliko su one već uređene da ne izvrši traženo sortiranje.

Sortiranje se može obaviti pomoću filesort metode ili pomoću indeksa, to možemo videti u Extra koloni rezultata EXPLAIN naredbe:

```
1 • use sakila;  
2 • SET optimizer_trace='enabled=on';  
3 • EXPLAIN SELECT title FROM film ORDER BY title;
```

Result Grid Filter Rows: <input type="text"/> Export: Wrap Cell Content:												
	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	film	NULL	index	NULL	idx_title	514	NULL	1000	100.00	Using index

Pošto kolona title tabele film ima svoj indeks, onda se koristi indeksiranje.



U upitima sledećeg oblika:

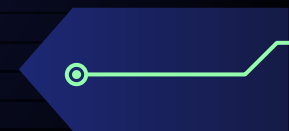

SELECT*FROM table1 ORDER BY key part1,key part2

Postojanje indeksa za key_part1 i key_part2 omogućava optimizatoru da izbegne sortiranje, ali pošto upit vrši selekciju svih kolona tabele table1, što nisu samo kolone key_part1 i key_part2, skeniranje celog indeksa i traženje vrsta tabele radi pronalaženja kolona koji nisu indeksirani može biti skuplje od skeniranja tabele i sortiranja rezultata. Ako je to slučaj, optimizator verovatno neće koristiti indeks.

Dve kolone u ORDER BY klauzuli mogu sortirati u istom smeru (obe ASC ili obe DESC), a mogu i u suprotnim smerovima (jedna ASC, druga DESC ili obrnuto):

SELECT*FROM table1 ORDER BY key part1 DESC,key part2 ASC

Optimizator može koristiti indeks nad key_part1 i key_part2 ako je key_part1 opadajući, a key_part2 rastući, ali može koristiti indeks na tim kolonama i ako je key_part1 rastući i key_part2 opadajući.



Slučajevi kada MySQL **ne može da koristi indekse** za razrešavanje ORDER BY klauzule, već koristi ***filesort***:

- Kada upit koristi ORDER BY nad različitim indeksima:

```
SELECT * FROM t1 ORDER BY key1, key2;
```

- Kada upit koristi ORDER BY nad neuzastopnim delovima indeksa:

```
SELECT * FROM t1 WHERE key2=constant ORDER BY key1_part1, key1_part3;
```

- Kada se indeks za pribavljanje redova razlikuje od indeksa koji se koristi za ORDER BY:

```
SELECT * FROM t1 WHERE key2=constant ORDER BY key1;
```

- Kada upit koristi ORDER BY sa izrazom koji sadrži izraze koji nisu naziv indeksa kolone:

```
SELECT * FROM t1 ORDER BY ABS(key);
```

```
SELECT * FROM t1 ORDER BY -key;
```

Optimizacija GROUP BY klauzule

Koraci koje preduzima MySQL optimizator

U ovom slučaju ili se kreira privremena tabela ili se koristi indeksiranje. U koloni Extra može se videti informacija o tome:

```
1 • use sakila;  
2 • EXPLAIN SELECT last_name FROM customer GROUP BY last_name;
```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:								Extra
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Using index
1	SIMPLE	customer	NULL	index	idx_last_name	idx_last_name	182	NULL	599	100.00	

```
1 • use sakila;  
2 • EXPLAIN SELECT email FROM customer GROUP BY email;
```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:								Extra
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Using temporary
1	SIMPLE	customer	NULL	ALL	NULL	NULL	NULL	NULL	599	100.00	



Optimizacija **DISTINCT** klauzule

Koraci koje preduzima MySQL optimizator



- Za DISTINCT u kombinaciji sa ORDER BY vrlo često je potrebno koristiti privremenu tabelu radi optimizacije upita.
- U većini slučajeva DISTINCT se može posmatrati kao poseban slučaj GROUP BY klauzule po sledećem pravilu:

Ukoliko nema WHERE ni LIMIT klauzule, postoji indeks nad kolonom koja je potrebna DISTINCT klauzuli i koriste se podaci iz samo jedne tabele, upit oblika:

SELECT DISTINCT column FROM table

Može se prevesti u upit oblika:

SELECT column FROM table GROUP BY column

Tako da su sledeća dva upita ekvivalentna:

SELECT DISTINCT col1, col2, col3 FROM table1 WHERE col1 > const;

SELECT col1, col2, col3 FROM table1 WHERE col1 > const GROUP BY col1, col2, col3;



Zbog ovoga se optimizacije koje koristimo za GROUP BY mogu primeniti i na DISTINCT klauzulu.

```
1 • use sakila;
2 • SET optimizer_trace="enabled=on";
3 • EXPLAIN SELECT DISTINCT last_name FROM customer;
4 • SELECT * FROM information_schema.optimizer_trace;
```

```
{
  "optimizing_distinct_group_by_order_by": {
    "changed_distinct_to_group_by": true,
    "simplifying_group_by": {
      "original_clause": "`customer`.`last_name`",
      "items": [
        {
          "item": "`customer`.`last_name`"
        }
      ],
      "resulting_clause_is_simple": true,
      "resulting_clause": "`customer`.`last_name`"
    }
  },
}
```



Optimizacija **LIMIT** klauzule

Koraci koje preduzima MySQL optimizator

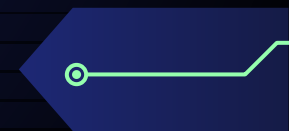



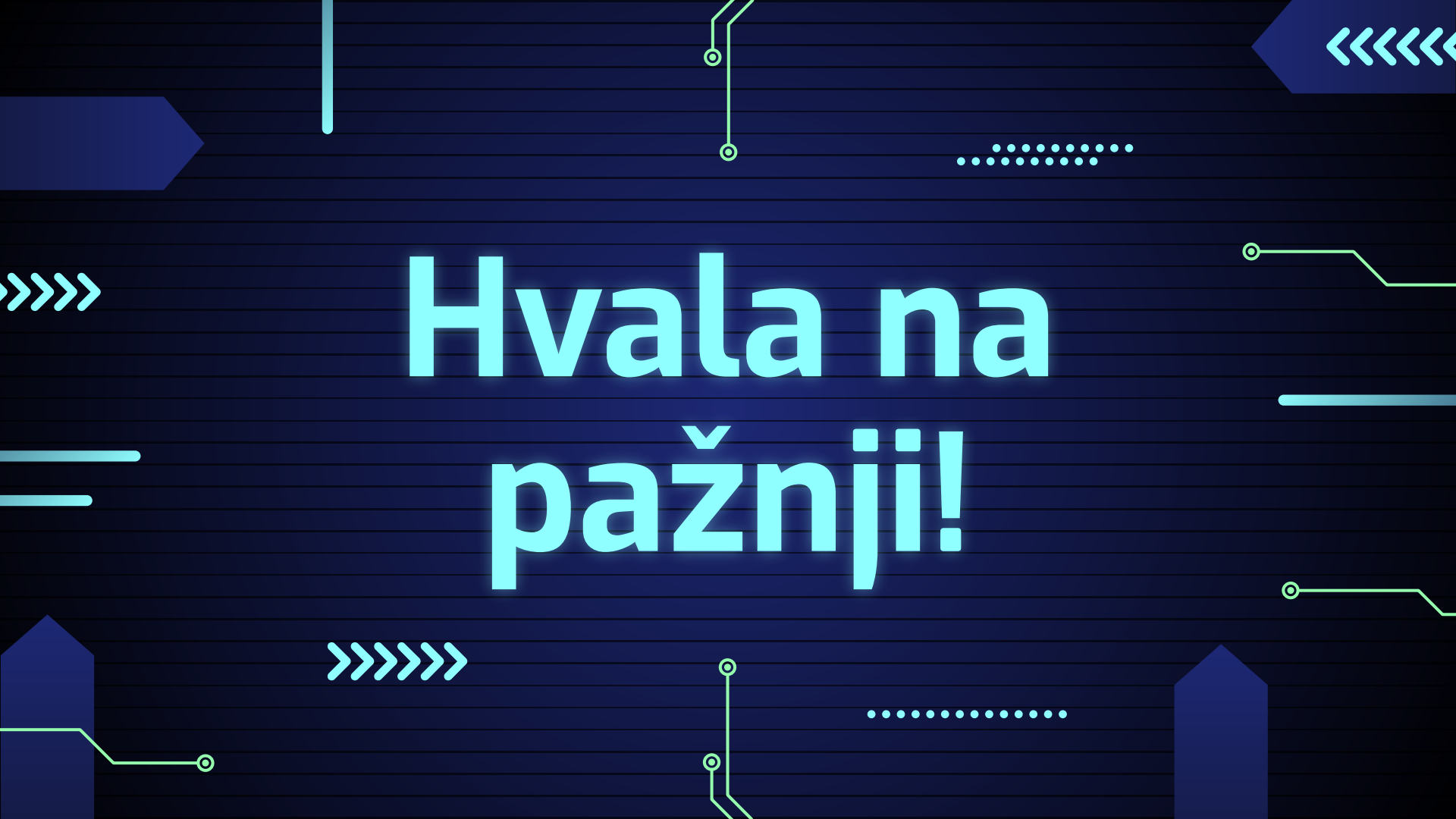
LIMIT klauzula se koristi kada u rezultatu nisu potrebne sve moguće vrste, već samo određeni broj vrsta. Kada se aktivira automatska optimizacija LIMIT upita, MySQL će izvršiti sledeće radnje:

1. Izvršiće se upit i naći se redovi koji odgovaraju uslovima upita.
2. Nakon toga, MySQL će koristiti tzv. "quick select" algoritam za sortiranje redova. Quick select je efikasan algoritam koji omogućava sortiranje samo prvih nekoliko redova, što je u skladu sa zahtevima klauzule LIMIT.
3. Nakon sortiranja redova, MySQL će vratiti samo prvih nekoliko redova koji su potrebni u skladu sa LIMIT klauzulom. Ostatak redova se neće vratiti, što čini izvršavanje upita bržim i efikasnijim.



Automatska optimizacija se vrši samo kada se koristi LIMIT klauzula sa određenim brojem redova koji se vraćaju. Ako se koristi LIMIT klauzula bez broja, automatska optimizacija se neće primeniti.



The background is a dark blue gradient. It features several abstract geometric elements: a large dark blue arrow pointing right in the top left; a series of white chevrons pointing left in the top right; a series of white chevrons pointing right in the middle left; a series of white chevrons pointing right in the bottom left; a series of white dots in the bottom right; and several thin white lines of varying lengths and orientations, some ending in small circles, scattered across the composition.

**Hvala na
pažnji!**