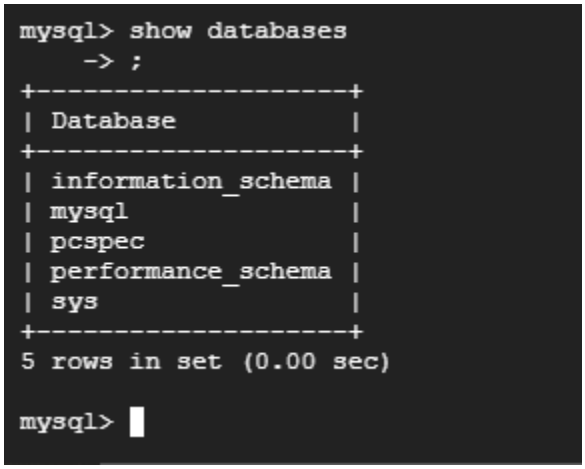
A screenshot of the connection (i.e. showing your terminal information)



DDL commands for your tables.

Product

```
CREATE TABLE Product(
        ProductId REAL PRIMARY KEY,
        BrandName varchar(30),
        ProductName varchar(30)
)
```

GPU

```
CREATE TABLE GPU(
        ProductId REAL PRIMARY KEY,
        GPU_Chip varchar(20),
        ReleaseYear INT,
        GPU_Memory varchar(30),
        GPU_Clock varchar(30),
        Memory_Clock varchar(30),
        foreign key (ProductId) references Product(ProductId)
        ON UPDATE CASCADE
);
```

CPU

```
CREATE TABLE CPU(
        ProductId REAL PRIMARY KEY,
        CodeName varchar(20),
```

```
        Cores varchar(10),
        Clock varchar(30),
        CPUSocket varchar(30),
        CPUProcess varchar(30),
        L3_Cache varchar(10),
        TDP varchar(10),
        ReleaseYear INT,
        foreign key (ProductId) references Product(ProductId)
        ON UPDATE CASCADE
);
```

MotherBoard

```
CREATE TABLE MotherBoard(
        ProductId REAL PRIMARY KEY,
        Socket varchar(30),
        Chipset varchar(30),
        SupportMemoryType varchar(30),
        foreign key (ProductId) references Product(ProductId)
        ON UPDATE CASCADE
);
```

Inserting at least 1000 rows in the tables. (Count Query)

This screenshot shows the Count() on CPU, Count() on GPU and Count() on Product

```
mysql> SELECT COUNT(*) FROM Product;
+----------+
| COUNT(*) |
+----------+
|     4628 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM CPU;
+----------+
| COUNT(*) |
+----------+
|     1031 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM GPU;
+----------+
| COUNT(*) |
+----------+
|     1473 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM MotherBoard;
+----------+
| COUNT(*) |
+----------+
|     2124 |
+----------+
1 row in set (0.01 sec)
```

Advanced Query 1 (Query itself, top 15 rolls, indexing)

The Ryzen 7 5700X series is a very popular CPU that many people are willing to use for their own PC. If we want to utilize this CPU, this query is able to show us all the motherboards and its detailed information that is compatible with the specific CPU.

```
11
12 •  select *
13    from  MotherBoard m natural join Product p
14    where Socket = (select c.CPUSocket
15                    from  CPU c natural join Product p
16                    where productName like "Ryzen 7 5700X%")
17    limit 15
18
19    -- select Count(*),c.CPUProcess
20    -- from  CPU c natural join Product p
21    -- where CPUSocket = (select m.Socket
22    --                    from MotherBoard m  natural join Product p
23    --                    where productName like "ASUS TUF Gaming X470 ATX%")
```

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

ProductId	Socket	Chipset	SupportMemoryType	BrandName	ProductName
31188	Socket AM4	X570	DDR4	AMD	Asus Prime X570 ATX
31189	Socket AM4	X570	DDR4	AMD	Asus Prime X570 Micro-ATX
31190	Socket AM4	X570	DDR4	AMD	Asus Prime X570 Mini-ITX
31191	Socket AM4	X570	DDR4	AMD	Asus TUF Gaming X570 ATX
31192	Socket AM4	X570	DDR4	AMD	Asus TUF Gaming X570 Micro-ATX
31193	Socket AM4	X570	DDR4	AMD	Asus TUF Gaming X570 Mini-ITX
31194	Socket AM4	X570	DDR4	AMD	Asus ROG Strix X570 ATX
31195	Socket AM4	X570	DDR4	AMD	Asus ROG Strix X570 Micro-ATX
31196	Socket AM4	X570	DDR4	AMD	Asus ROG Strix X570 Mini-ITX
31197	Socket AM4	X570	DDR4	AMD	Asus ROG Maximus X570 ATX
31198	Socket AM4	X570	DDR4	AMD	Asus ROG Maximus X570 Micro-AT
31199	Socket AM4	X570	DDR4	AMD	Asus ROG Maximus X570 Mini-ITX
31200	Socket AM4	X570	DDR4	AMD	Gigabyte Aorus XTREME X570 ATX
31201	Socket AM4	X570	DDR4	AMD	Gigabyte Aorus XTREME X570 Mic
31202	Socket AM4	X570	DDR4	AMD	Gigabyte Aorus XTREME X570 Min

Execution of query 1 without any index

```
mysql> EXPLAIN ANALYZE
    -> select *
    -> from MotherBoard m natural join Product p
    -> where Socket = (select c.CPUSocket from  CPU c natural join Product p where productName like "Ryzen 7 5700X%")
    -> limit 15;
+--------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                |
+--------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=288.99 rows=15) (actual time=2.215..2.237 rows=15 loops=1)
    -> Nested loop inner join  (cost=288.99 rows=212) (actual time=2.213..2.235 rows=15 loops=1)
        -> Filter: (m.`Socket` = (select #2))  (cost=214.65 rows=212) (actual time=2.208..2.214 rows=15 loops=1)
            -> Table scan on m  (cost=214.65 rows=2124) (actual time=0.049..0.373 rows=1203 loops=1)
            -> Select #2 (subquery in condition; run only once)
                -> Nested loop inner join  (cost=465.95 rows=115) (actual time=1.138..1.694 rows=1 loops=1)
                    -> Table scan on c  (cost=105.10 rows=1031) (actual time=0.034..0.317 rows=1031 loops=1)
                    -> Filter: (p.ProductName like 'Ryzen 7 5700X%')  (cost=0.25 rows=0) (actual time=0.001..0.001 rows=0 loops=1031)
                        -> Single-row index lookup on p using PRIMARY (ProductId=c.ProductId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1031)
        -> Single-row index lookup on p using PRIMARY (ProductId=m.ProductId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=15)
 |
+--------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

1. Execution of query 1 with index on CPU (CPUSocket)

```
mysql> EXPLAIN ANALYZE
    -> select *
    -> from MotherBoard m natural join Product p
    -> where Socket = (select c.CPUSocket from  CPU c natural join Product p where productName like "Ryzen 7 5700X%")
    -> limit 15;
+--------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                |
+--------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=288.99 rows=15) (actual time=2.327..2.350 rows=15 loops=1)
    -> Nested loop inner join  (cost=288.99 rows=212) (actual time=2.326..2.348 rows=15 loops=1)
        -> Filter: (m.`Socket` = (select #2))  (cost=214.65 rows=212) (actual time=2.320..2.326 rows=15 loops=1)
            -> Table scan on m  (cost=214.65 rows=2124) (actual time=0.049..0.372 rows=1203 loops=1)
            -> Select #2 (subquery in condition; run only once)
                -> Nested loop inner join  (cost=465.95 rows=115) (actual time=1.269..1.805 rows=1 loops=1)
                    -> Index scan on c using CPU_S  (cost=105.10 rows=1031) (actual time=0.195..0.423 rows=1031 loops=1)
                    -> Filter: (p.ProductName like 'Ryzen 7 5700X%')  (cost=0.25 rows=0) (actual time=0.001..0.001 rows=0 loops=1031)
                        -> Single-row index lookup on p using PRIMARY (ProductId=c.ProductId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1031)
        -> Single-row index lookup on p using PRIMARY (ProductId=m.ProductId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=15)
 |
+--------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

The main idea of inducing index on CPU socket is because we need to select cpu socket from the CPU table to find the socket with product name “Ryzen 7 5700X%”. This experiment shows that inducing index on CPU socket will not gain any profit on this query. We believe this happens because we are not trying to use socket as an attribute to search for any information, but to find the socket with other attributes.

2. Execution of query 1 with index on CPU (CPUSocket) and MotherBoard (Socket)

```
mysql> EXPLAIN ANALYZE
    -> select *
    -> from MotherBoard m natural join Product p
    -> where Socket = (select c.CPUSocket from  CPU c natural join Product p where productName like "Ryzen 7 5700X%")
    -> limit 15;
+-----------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                       |
+-----------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=136.35 rows=15) (actual time=0.211..0.234 rows=15 loops=1)
    -> Nested loop inner join  (cost=136.35 rows=288) (actual time=0.211..0.232 rows=15 loops=1)
        -> Filter: (m.`Socket` = (select #2))  (cost=35.55 rows=288) (actual time=0.205..0.212 rows=15 loops=1)
            -> Index lookup on m using MotherBoard_S (Socket=(select #2))  (cost=35.55 rows=288) (actual time=0.204..0.206 rows=15 loops=1)
            -> Select #2 (subquery in condition; run only once)
                -> Nested loop inner join  (cost=465.95 rows=115) (actual time=1.197..1.726 rows=1 loops=1)
                    -> Index scan on c using CPU_S  (cost=105.10 rows=1031) (actual time=0.047..0.295 rows=1031 loops=1)
                    -> Filter: (p.ProductName like 'Ryzen 7 5700X%')  (cost=0.25 rows=0) (actual time=0.001..0.001 rows=0 loops=1031)
                        -> Single-row index lookup on p using PRIMARY (ProductId=c.ProductId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1031)
        -> Single-row index lookup on p using PRIMARY (ProductId=m.ProductId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=15)
 |
+-----------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

The main idea of inducing an index on MotherBoard Socket is because we need to find Motherboards that have the same socket as the CPU named "Ryzen 7 5700X%". We believed this index should be useful because indeed we are trying to find some motherboards with required socket type. This index increases the performance of this query as we expected. We kept the index on CPU Socket because we believed it will not affect the experimental result in this case and it is used in another experiment as well. Later we deleted the index on CPU Socket, and only tested the index on MotherBoard; the cost reduced, and the result is the same as the above plot.

3. Execution of query 1 with index on Product (productName)

```
mysql> CREATE INDEX Product_N ON Product (pro);
ERROR 1072 (42000): Key column 'pro' doesn't exist in table
mysql> CREATE INDEX Product_N ON Product (productName);
Query OK, 0 rows affected (0.21 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE
    -> select *
    -> from MotherBoard m natural join Product p
    -> where Socket = (select c.CPUSocket from  CPU c natural join Product p where productName like "Ryzen 7 5700X%")
    -> limit 15;
+-----------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                       |
+-----------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=288.99 rows=15) (actual time=0.533..0.560 rows=15 loops=1)
    -> Nested loop inner join  (cost=288.99 rows=212) (actual time=0.532..0.558 rows=15 loops=1)
        -> Filter: (m.`Socket` = (select #2))  (cost=214.65 rows=212) (actual time=0.527..0.533 rows=15 loops=1)
            -> Table scan on m  (cost=214.65 rows=2124) (actual time=0.033..0.359 rows=1203 loops=1)
            -> Select #2 (subquery in condition; run only once)
                -> Nested loop inner join  (cost=1.49 rows=1) (actual time=0.020..0.032 rows=1 loops=1)
                    -> Filter: (p.ProductName like 'Ryzen 7 5700X%')  (cost=1.14 rows=1) (actual time=0.012..0.024 rows=1 loops=1)
                        -> Index range scan on p using Product_N  (cost=1.14 rows=1) (actual time=0.009..0.021 rows=1 loops=1)
                    -> Single-row index lookup on c using PRIMARY (ProductId=p.ProductId)  (cost=0.35 rows=1) (actual time=0.007..0.007 rows=1 loops=1)
        -> Single-row index lookup on p using PRIMARY (ProductId=m.ProductId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=15)
 |
+-----------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

The main idea of inducing an index on Product name is because we need to find the CPU named “Ryzen 7 5700X%” in our subquery. We believed this index should be useful because indeed we are trying to find some motherboards with required socket type. This index increases the performance of this query as we expected. We kept the index on CPU Socket because we believed it will not affect the experimental result in this case and it is used in another experiment as well. Later we deleted the index on CPU Socket, and only tested the index on MotherBoard; the cost reduced, and the result is the same as the above plot.

Advanced Query 2 (Query itself, top 15 rolls, indexing)

We design this query because sometimes we have a specific type of motherboard, and we want to know how many different kinds of CPU in total we can choose based on the CPU process. So, this query shows different types of CPU Process groups and the count of each group.

```sql
select Count(*),c.CPUProcess
from  CPU c natural join Product p
where CPUSocket = (select m.Socket
                   from MotherBoard m  natural join Product p
                   where productName like "ASUS TUF Gaming X470 ATX%")
group by c.CPUProcess
```

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

Count(*)	CPUProcess
45	7 nm
22	12 nm
25	28 nm
24	14 nm

This screenshot shows without create index (default index) for query 2

```
mysql> EXPLAIN ANALYZE
    -> select Count(*),c.CPUProcess
    -> from  CPU c natural join Product p
    -> where CPUSocket = (select m.Socket from MotherBoard m  natural join Product p where productName like "ASUS TUF Gaming X470 ATX%")
    -> group by c.CPUProcess;
+------------------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                                                              |
+------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=0.001..0.001 rows=4 loops=1)
    -> Aggregate using temporary table  (actual time=2.537..2.537 rows=4 loops=1)
        -> Nested loop inner join  (cost=141.19 rows=103) (actual time=2.093..2.447 rows=116 loops=1)
            -> Filter: (c.CPUSocket = (select #2))  (cost=105.10 rows=103) (actual time=2.086..2.295 rows=116 loops=1)
                -> Table scan on c  (cost=105.10 rows=1031) (actual time=0.053..0.359 rows=1031 loops=1)
                -> Select #2 (subquery in condition; run only once)
                    -> Nested loop inner join  (cost=647.51 rows=514) (actual time=1.411..1.726 rows=1 loops=1)
                        -> Filter: (p.ProductName like 'ASUS TUF Gaming X470 ATX%')  (cost=467.55 rows=514) (actual time=1.390..1.705 rows=1 loops=1)
                            -> Table scan on p  (cost=467.55 rows=4628) (actual time=0.024..1.196 rows=4628 loops=1)
                        -> Single-row index lookup on m using PRIMARY (ProductId=p.ProductId)  (cost=0.25 rows=1) (actual time=0.019..0.019 rows=1 loops=1)
            -> Single-row index lookup on p using PRIMARY (ProductId=c.ProductId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=116)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

1. Execution of query 2 with index on CPU (CPUSocket)

```
mysql> CREATE INDEX CPU_S ON CPU (CPUSocket);
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE
    -> select Count(*),c.CPUProcess
    -> from  CPU c natural join Product p
    -> where CPUSocket = (select m.Socket from MotherBoard m  natural join Product p where productName like "ASUS TUF Gaming X470 ATX%")
    -> group by c.CPUProcess;
+-------------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                  |
+-------------------------------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=0.001..0.001 rows=4 loops=1)
    -> Aggregate using temporary table  (actual time=0.646..0.647 rows=4 loops=1)
        -> Nested loop inner join  (cost=58.20 rows=116) (actual time=0.328..0.555 rows=116 loops=1)
            -> Filter: (c.CPUSocket = (select #2))  (cost=17.60 rows=116) (actual time=0.319..0.402 rows=116 loops=1)
                -> Index lookup on c using CPU_S (CPUSocket=(select #2))  (cost=17.60 rows=116) (actual time=0.318..0.363 rows=116 loops=1)
                -> Select #2 (subquery in condition; run only once)
                    -> Nested loop inner join  (cost=647.51 rows=514) (actual time=1.351..1.670 rows=1 loops=1)
                        -> Filter: (p.ProductName like 'ASUS TUF Gaming X470 ATX%')  (cost=467.55 rows=514) (actual time=1.333..1.652 rows=1 loops=1)
                            -> Table scan on p  (cost=467.55 rows=4628) (actual time=0.045..1.169 rows=4628 loops=1)
                        -> Single-row index lookup on m using PRIMARY (ProductId=p.ProductId)  (cost=0.25 rows=1) (actual time=0.016..0.016 rows=1 loops=1)
            -> Single-row index lookup on p using PRIMARY (ProductId=c.ProductId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=116)
 |
+-------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

The main idea of inducing index on CPU socket is because we need to find the CPU with the same socket type as the motherboard we choose, the socket type is used as an attribute to find required CPU. This experiment shows that inducing index on CPU socket will dramatically increase the performance of this query.

2. Execution of query 1 with index on CPU (CPUSocket) and MotherBoard (Socket)

```
mysql> CREATE INDEX MotherBoard_S ON MotherBoard (Socket);
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE
    -> select Count(*),c.CPUProcess
    -> from  CPU c natural join Product p
    -> where CPUSocket = (select m.Socket from MotherBoard m  natural join Product p where productName like "ASUS TUF Gaming X470 ATX%")
    -> group by c.CPUProcess;
+-------------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                  |
+-------------------------------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=0.001..0.001 rows=4 loops=1)
    -> Aggregate using temporary table  (actual time=1.457..1.458 rows=4 loops=1)
        -> Nested loop inner join  (cost=58.20 rows=116) (actual time=1.162..1.363 rows=116 loops=1)
            -> Filter: (c.CPUSocket = (select #2))  (cost=17.60 rows=116) (actual time=1.151..1.224 rows=116 loops=1)
                -> Index lookup on c using CPU_S (CPUSocket=(select #2))  (cost=17.60 rows=116) (actual time=1.147..1.182 rows=116 loops=1)
                -> Select #2 (subquery in condition; run only once)
                    -> Nested loop inner join  (cost=647.51 rows=514) (actual time=1.350..1.672 rows=1 loops=1)
                        -> Filter: (p.ProductName like 'ASUS TUF Gaming X470 ATX%')  (cost=467.55 rows=514) (actual time=1.329..1.650 rows=1 loops=1)
                            -> Table scan on p  (cost=467.55 rows=4628) (actual time=0.045..1.169 rows=4628 loops=1)
                        -> Single-row index lookup on m using PRIMARY (ProductId=p.ProductId)  (cost=0.25 rows=1) (actual time=0.019..0.019 rows=1 loops=1)
            -> Single-row index lookup on p using PRIMARY (ProductId=c.ProductId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=116)
 |
+-------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

The main idea of inducing index on MotherBoard Socket is because MotherBoard Socket is selected in our subquery. This experiment shows that inducing index on MotherBoard Socket will not gain any profit on this query. We believe this happens because we are not trying to use socket as an attribute to search for any information, but to find the corresponding socket type of MotherBoard with ProductName.

3. Execution of query 2 with index on CPU (CPUProcess)

```
mysql> CREATE INDEX CPU_P ON CPU (CPUProcess);
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE
    -> select Count(*),c.CPUProcess
    -> from  CPU c natural join Product p
    -> where CPUSocket = (select m.Socket from MotherBoard m  natural join Product p where productName like "ASUS TUF Gaming X470 ATX%")
    -> group by c.CPUProcess;
+--------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                |
+--------------------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=0.001..0.001 rows=4 loops=1)
    -> Aggregate using temporary table  (actual time=2.685..2.686 rows=4 loops=1)
        -> Nested loop inner join  (cost=141.19 rows=103) (actual time=2.230..2.588 rows=116 loops=1)
            -> Filter: (c.CPUSocket = (select #2))  (cost=105.10 rows=103) (actual time=2.221..2.429 rows=116 loops=1)
                -> Table scan on c  (cost=105.10 rows=1031) (actual time=0.058..0.365 rows=1031 loops=1)
                -> Select #2 (subquery in condition; run only once)
                    -> Nested loop inner join  (cost=647.51 rows=514) (actual time=1.484..1.839 rows=1 loops=1)
                        -> Filter: (p.ProductName like 'ASUS TUF Gaming X470 ATX%')  (cost=467.55 rows=514) (actual time=1.460..1.814 rows=1 loops=1)
                            -> Table scan on p  (cost=467.55 rows=4628) (actual time=0.029..1.311 rows=4628 loops=1)
                        -> Single-row index lookup on m using PRIMARY (ProductId=p.ProductId)  (cost=0.25 rows=1) (actual time=0.022..0.022 rows=1 loops=1)
            -> Single-row index lookup on p using PRIMARY (ProductId=c.ProductId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=116)
 |
+--------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

The main idea of inducing index on CPU Process is because CPU Process is selected and grouped by in our query. This experiment shows that inducing index on CPU Process will not gain any profit on this query. We believe this happens because CPU Process is not used as an attribute to search for any information in this query.