RocksDB on Ubuntu

Quick Start Guide

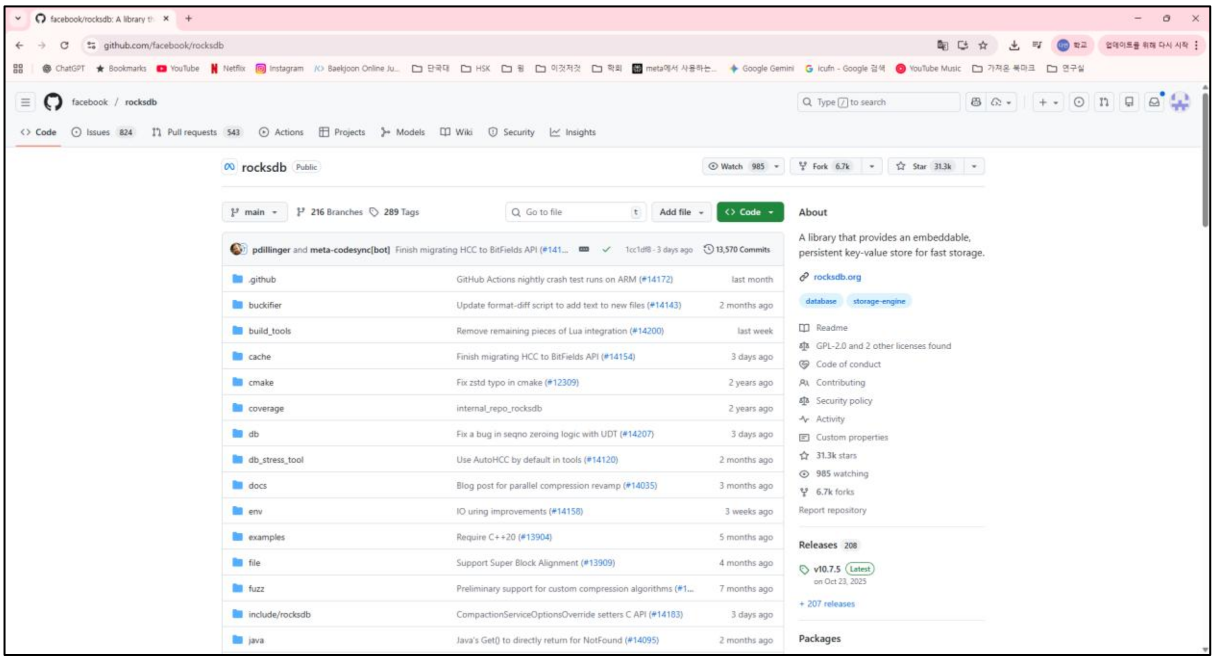
Dankook Univ. System Software Lab

Dayeon Wee, Yongmin Lee

목차

1. RocksDB 설치

https://github.com/facebook/rocksdb



rocksdb Public

pdillinger and meta-codesync[bot] Finish migrating HCC to BitFields API (#141... 1cc1df8 · 3 days ago 13,570 Commits

.github	GitHub Actions nightly crash test runs on ARM (#14172)	last month
buckifier	Update format-diff script to add text to new files (#14143)	2 months ago
build_tools	Remove remaining pieces of Lua integration (#14200)	last week
cache	Finish migrating HCC to BitFields API (#14154)	3 days ago
cmake	Fix zstd typo in cmake (#12309)	2 years ago
coverage	internal_repo_rocksdb	2 years ago
db	Fix a bug in seqno zeroing logic with UDT (#14207)	3 days ago
db_stress_tool	Use AutoHCC by default in tools (#14120)	2 months ago
docs	Blog post for parallel compression revamp (#14035)	3 months ago
env	IO uring improvements (#14158)	3 weeks ago
examples	Require C++20 (#13904)	5 months ago
file	Support Super Block Alignment (#13909)	4 months ago
fuzz	Preliminary support for custom compression algorithms (#1...	7 months ago
include/rocksdb	CompactionServiceOptionsOverride setters C API (#14183)	3 days ago
java	Java's Get() to directly return for NotFound (#14095)	2 months ago

About

A library that provides an embeddable, persistent key-value store for fast storage.

rocksdb.org

database storage-engine

Readme
GPL-2.0 and 2 other licenses found
Code of conduct
Contributing
Security policy
Activity
Custom properties
31.3k stars
985 watching
6.7k forks
Report repository

Releases 208

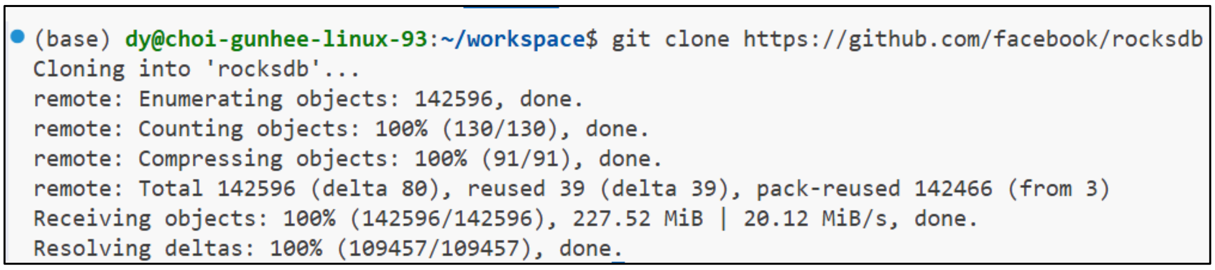
v10.7.5 Latest
on Oct 23, 2025

+ 207 releases

Packages

우분투 환경에서

git clone https://github.com/facebook/rocksdb

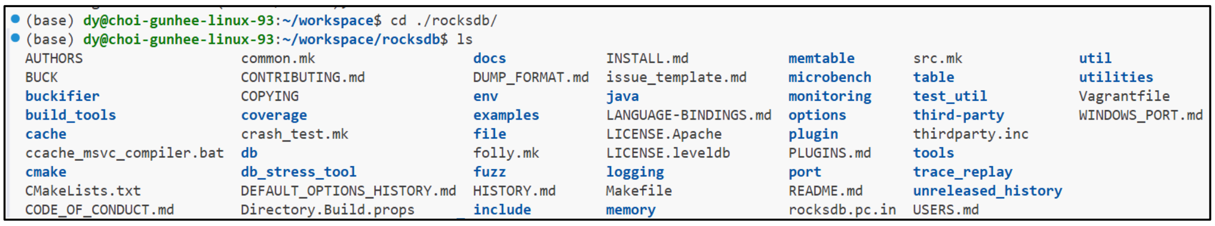


```
(base) dy@choi-gunhee-linux-93:~/workspace$ git clone https://github.com/facebook/rocksdb
Cloning into 'rocksdb'...
remote: Enumerating objects: 142596, done.
remote: Counting objects: 100% (130/130), done.
remote: Compressing objects: 100% (91/91), done.
remote: Total 142596 (delta 80), reused 39 (delta 39), pack-reused 142466 (from 3)
Receiving objects: 100% (142596/142596), 227.52 MiB | 20.12 MiB/s, done.
Resolving deltas: 100% (109457/109457), done.
```

cd RocksDB

ls

RocksDB 내부 파일들이 위치해있음



```
(base) dy@choi-gunhee-linux-93:~/workspace$ cd ./rocksdb/
(base) dy@choi-gunhee-linux-93:~/workspace/rocksdb$ ls
AUTHORS                    common.mk                  docs              INSTALL.md               memtable        src.mk              util
BUCK                       CONTRIBUTING.md            DUMP_FORMAT.md    issue_template.md        microbench      table               utilities
buckifier                  COPYING                    env               java                     monitoring      test_util           Vagrantfile
build_tools                coverage                   examples          LANGUAGE-BINDINGS.md     options         third-party         WINDOWS_PORT.md
cache                      crash_test.mk              file              LICENSE.Apache           plugin          thirdparty.inc
ccache_msvc_compiler.bat   db                         folly.mk          LICENSE.leveldb          PLUGINS.md      tools
cmake                      db_stress_tool             fuzz              logging                  port            trace_replay
CMakeLists.txt             DEFAULT_OPTIONS_HISTORY.md HISTORY.md        Makefile                 README.md       unreleased_history
CODE_OF_CONDUCT.md         Directory.Build.props      include           memory                   rocksdb.pc.in   USERS.md
```

2. RocksDB 빌드

2.1 의존성 설치

sudo apt-get install libgflags-dev

sudo apt-get install libsnappy-dev

sudo apt-get install zlib1g-dev

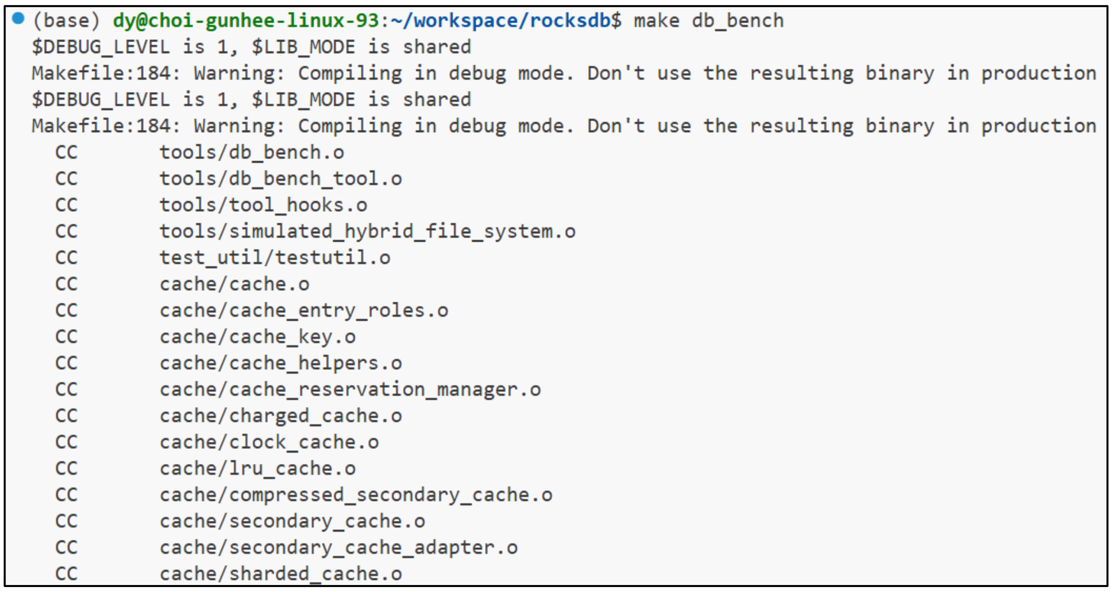
sudo apt-get install libbz2-dev

sudo apt-get install liblz4-dev

sudo apt-get install libzstd-dev

2.2 빌드

make db_bench



```
● (base) dy@choi-gunhee-linux-93:~/workspace/rocksdb$ make db_bench
  $DEBUG_LEVEL is 1, $LIB_MODE is shared
  Makefile:184: Warning: Compiling in debug mode. Don't use the resulting binary in production
  $DEBUG_LEVEL is 1, $LIB_MODE is shared
  Makefile:184: Warning: Compiling in debug mode. Don't use the resulting binary in production
    CC       tools/db_bench.o
    CC       tools/db_bench_tool.o
    CC       tools/tool_hooks.o
    CC       tools/simulated_hybrid_file_system.o
    CC       test_util/testutil.o
    CC       cache/cache.o
    CC       cache/cache_entry_roles.o
    CC       cache/cache_key.o
    CC       cache/cache_helpers.o
    CC       cache/cache_reservation_manager.o
    CC       cache/charged_cache.o
    CC       cache/clock_cache.o
    CC       cache/lru_cache.o
    CC       cache/compressed_secondary_cache.o
    CC       cache/secondary_cache.o
    CC       cache/secondary_cache_adapter.o
    CC       cache/sharded_cache.o
```

빌드 시 약 5~10분 소요

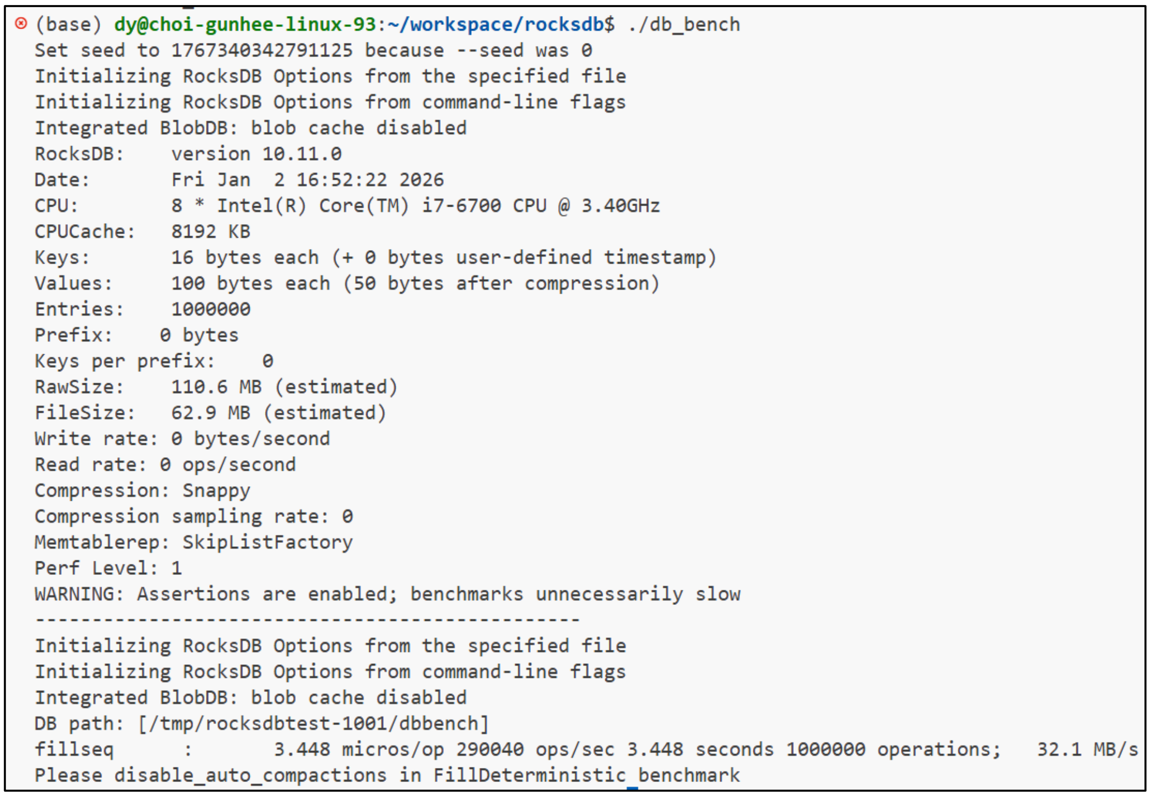
3. db_bench 실행

- db_bench는 RocksDB에 포함된 성능 벤치마크 및 스트레스 테스트 도구

- 다양한 워크로드를 실행해 지연시간·처리량·쓰기 증폭을 측정

- RocksDB 설정 옵션을 바꿔가며 튜닝 효과를 정량적으로 비교 가능

./db_bench를 모든 옵션을 디폴트로 실행했을 때의 출력 결과

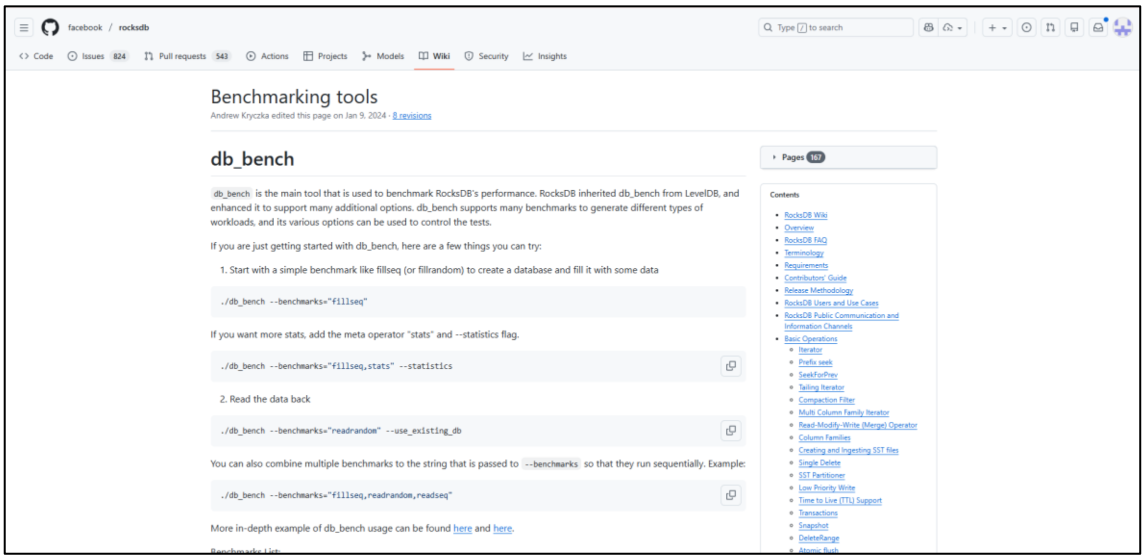


```
⊗ (base) dy@choi-gunhee-linux-93:~/workspace/rocksdb$ ./db_bench
  Set seed to 1767340342791125 because --seed was 0
  Initializing RocksDB Options from the specified file
  Initializing RocksDB Options from command-line flags
  Integrated BlobDB: blob cache disabled
  RocksDB:    version 10.11.0
  Date:       Fri Jan  2 16:52:22 2026
  CPU:        8 * Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz
  CPUCache:   8192 KB
  Keys:       16 bytes each (+ 0 bytes user-defined timestamp)
  Values:     100 bytes each (50 bytes after compression)
  Entries:    1000000
  Prefix:    0 bytes
  Keys per prefix:    0
  RawSize:    110.6 MB (estimated)
  FileSize:   62.9 MB (estimated)
  Write rate: 0 bytes/second
  Read rate: 0 ops/second
  Compression: Snappy
  Compression sampling rate: 0
  Memtablerep: SkipListFactory
  Perf Level: 1
  WARNING: Assertions are enabled; benchmarks unnecessarily slow
  ------------------------------------------------
  Initializing RocksDB Options from the specified file
  Initializing RocksDB Options from command-line flags
  Integrated BlobDB: blob cache disabled
  DB path: [/tmp/rocksdbtest-1001/dbbench]
  fillseq      :       3.448 micros/op 290040 ops/sec 3.448 seconds 1000000 operations;   32.1 MB/s
  Please disable_auto_compactions in FillDeterministic benchmark
```

3.1 db_bench 옵션

기본적인 옵션들은 위키에 있음

https://github.com/facebook/rocksdb/wiki/Benchmarking-tools



Benchmarking tools

Andrew Kryczka edited this page on Jan 9, 2024 · 8 revisions

db_bench

db_bench is the main tool that is used to benchmark RocksDB's performance. RocksDB inherited db_bench from LevelDB, and enhanced it to support many additional options. db_bench supports many benchmarks to generate different types of workloads, and its various options can be used to control the tests.

If you are just getting started with db_bench, here are a few things you can try:

1. Start with a simple benchmark like fillseq (or fillrandom) to create a database and fill it with some data

```
./db_bench --benchmarks="fillseq"
```

If you want more stats, add the meta operator "stats" and --statistics flag.

```
./db_bench --benchmarks="fillseq,stats" --statistics
```

2. Read the data back

```
./db_bench --benchmarks="readrandom" --use_existing_db
```

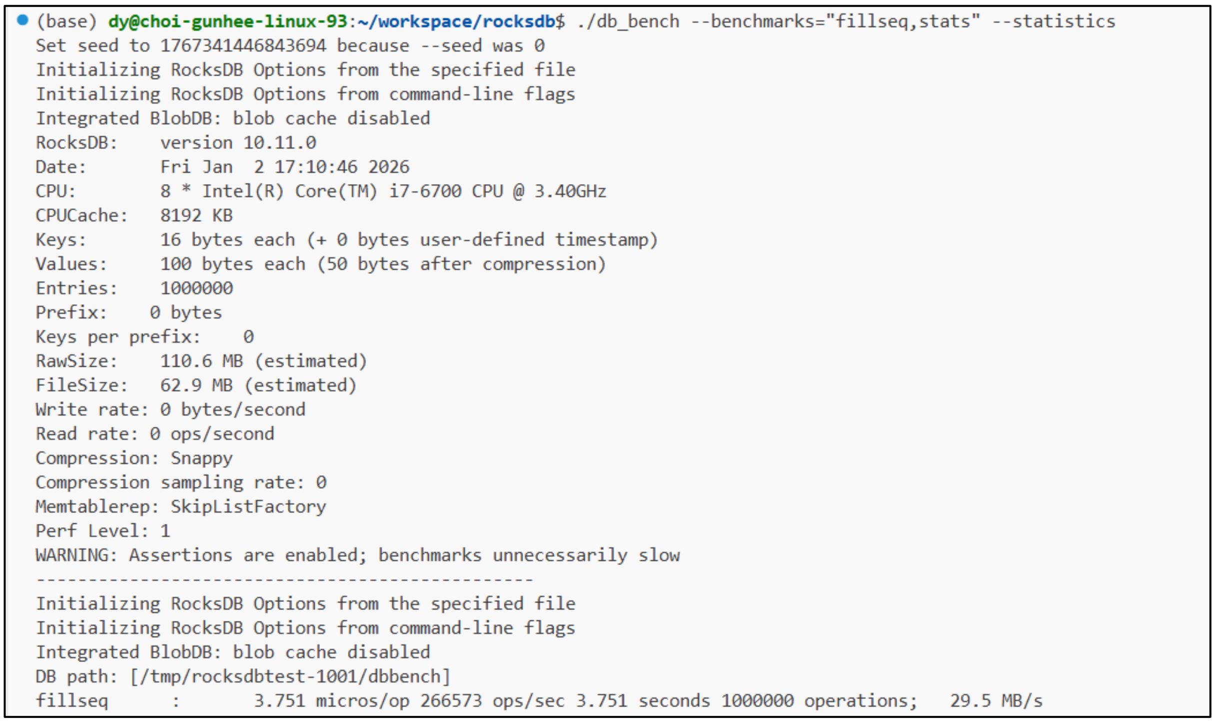
You can also combine multiple benchmarks to the string that is passed to `--benchmarks` so that they run sequentially. Example:

```
./db_bench --benchmarks="fillseq,readrandom,readseq"
```

More in-depth example of db_bench usage can be found here and here.

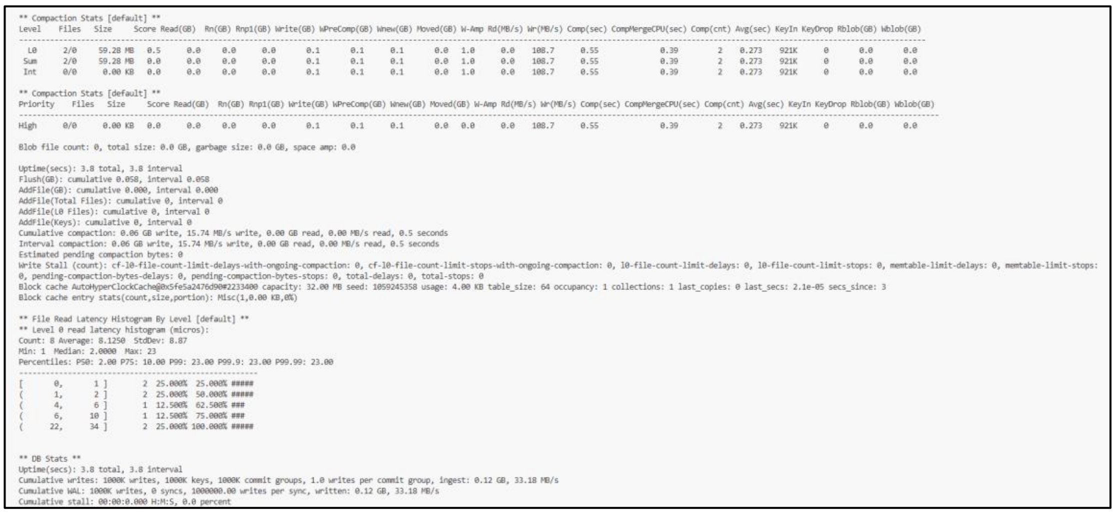
예를 들어,

./db_bench –benchmarks=”fillseq,stats” –statistics 와 같이 실행하게 되면

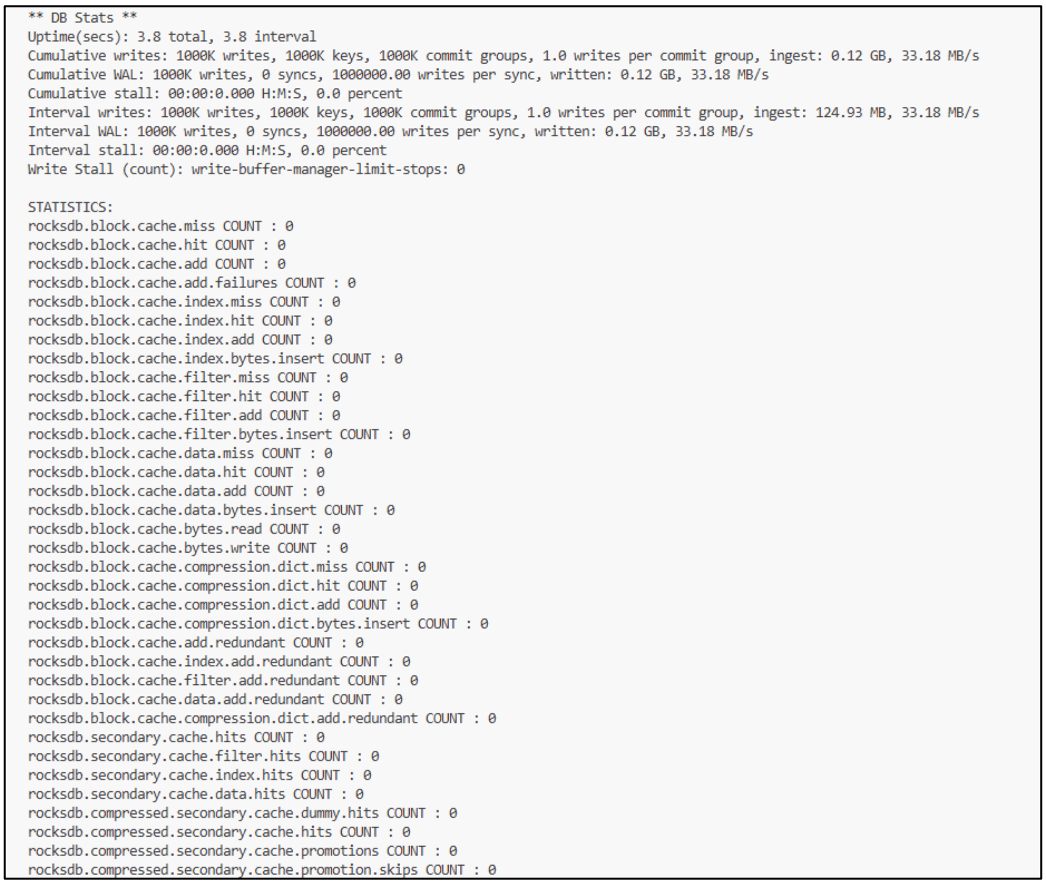


```
(base) dy@choi-gunhee-linux-93:~/workspace/rocksdb$ ./db_bench --benchmarks="fillseq,stats" --statistics
Set seed to 1767341446843694 because --seed was 0
Initializing RocksDB Options from the specified file
Initializing RocksDB Options from command-line flags
Integrated BlobDB: blob cache disabled
RocksDB:    version 10.11.0
Date:       Fri Jan  2 17:10:46 2026
CPU:        8 * Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz
CPUCache:   8192 KB
Keys:       16 bytes each (+ 0 bytes user-defined timestamp)
Values:     100 bytes each (50 bytes after compression)
Entries:    1000000
Prefix:    0 bytes
Keys per prefix:    0
RawSize:    110.6 MB (estimated)
FileSize:   62.9 MB (estimated)
Write rate: 0 bytes/second
Read rate: 0 ops/second
Compression: Snappy
Compression sampling rate: 0
Memtablerep: SkipListFactory
Perf Level: 1
WARNING: Assertions are enabled; benchmarks unnecessarily slow
------------------------------------------------
Initializing RocksDB Options from the specified file
Initializing RocksDB Options from command-line flags
Integrated BlobDB: blob cache disabled
DB path: [/tmp/rocksdbtest-1001/dbbench]
fillseq      :       3.751 micros/op 266573 ops/sec 3.751 seconds 1000000 operations;   29.5 MB/s
```

간략한 실행 결과와



```
** Compaction Stats [default] **
Level    Files   Size     Score Read(GB)  Rn(GB) Rnp1(GB) Write(GB) WPreComp(GB) Wnew(GB) Moved(GB) W-Amp Rd(MB/s) Wr(MB/s) Comp(sec) CompMergeCPU(sec) Comp(cnt) Avg(sec) KeyIn KeyDrop Rblob(GB) Wblob(GB)
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
  L0      2/0    59.28 MB   0.5      0.0     0.0      0.0       0.1       0.1       0.1       0.0   1.0      0.0    108.7      0.55              0.39         2    0.273   921K      0       0.0       0.0
 Sum      2/0    59.28 MB   0.0      0.0     0.0      0.0       0.1       0.1       0.1       0.0   1.0      0.0    108.7      0.55              0.39         2    0.273   921K      0       0.0       0.0
 Int      0/0     0.00 KB   0.0      0.0     0.0      0.0       0.1       0.1       0.1       0.0   1.0      0.0    108.7      0.55              0.39         2    0.273   921K      0       0.0       0.0

** Compaction Stats [default] **
Priority    Files   Size     Score Read(GB)  Rn(GB) Rnp1(GB) Write(GB) WPreComp(GB) Wnew(GB) Moved(GB) W-Amp Rd(MB/s) Wr(MB/s) Comp(sec) CompMergeCPU(sec) Comp(cnt) Avg(sec) KeyIn KeyDrop Rblob(GB) Wblob(GB)
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
High      0/0     0.00 KB   0.0      0.0     0.0      0.0       0.1       0.1       0.1       0.0   0.0      0.0    108.7      0.55              0.39         2    0.273   921K      0       0.0       0.0

Blob file count: 0, total size: 0.0 GB, garbage size: 0.0 GB, space amp: 0.0

Uptime(secs): 3.8 total, 3.8 interval
Flush(GB): cumulative 0.058, interval 0.058
AddFile(GB): cumulative 0.000, interval 0.000
AddFile(Total Files): cumulative 0, interval 0
AddFile(L0 Files): cumulative 0, interval 0
AddFile(Keys): cumulative 0, interval 0
Cumulative compaction: 0.06 GB write, 15.74 MB/s write, 0.00 GB read, 0.00 MB/s read, 0.5 seconds
Interval compaction: 0.06 GB write, 15.74 MB/s write, 0.00 GB read, 0.00 MB/s read, 0.5 seconds
Estimated pending compaction bytes: 0
Write Stall (count): cf-l0-file-count-limit-delays-with-ongoing-compaction: 0, cf-l0-file-count-limit-stops-with-ongoing-compaction: 0, l0-file-count-limit-delays: 0, l0-file-count-limit-stops: 0, memtable-limit-delays: 0, memtable-limit-stops:
0, pending-compaction-bytes-delays: 0, pending-compaction-bytes-stops: 0, total-delays: 0, total-stops: 0
Block cache AutoHyperClockCache@0x5fe5a2476d90#2233400 capacity: 32.00 MB seed: 1059245358 usage: 4.00 KB table_size: 64 occupancy: 1 collections: 1 last_copies: 0 last_secs: 2.1e-05 secs_since: 3
Block cache entry stats(count,size,portion): Misc(1,0.00 KB,0%)

** File Read Latency Histogram By Level [default] **
** Level 0 read latency histogram (micros):
Count: 8 Average: 8.1250  StdDev: 8.87
Min: 1  Median: 2.0000  Max: 23
Percentiles: P50: 2.00 P75: 10.00 P99: 23.00 P99.9: 23.00 P99.99: 23.00
------------------------------------------------------
[       0,       1 ]        2  25.000%  25.000% #####
(       1,       2 ]        2  25.000%  50.000% #####
(       4,       6 ]        1  12.500%  62.500% ###
(       6,      10 ]        1  12.500%  75.000% ###
(      22,      34 ]        2  25.000% 100.000% #####


** DB Stats **
Uptime(secs): 3.8 total, 3.8 interval
Cumulative writes: 1000K writes, 1000K keys, 1000K commit groups, 1.0 writes per commit group, ingest: 0.12 GB, 33.18 MB/s
Cumulative WAL: 1000K writes, 0 syncs, 1000000.00 writes per sync, written: 0.12 GB, 33.18 MB/s
Cumulative stall: 00:00:0.000 H:M:S, 0.0 percent
```

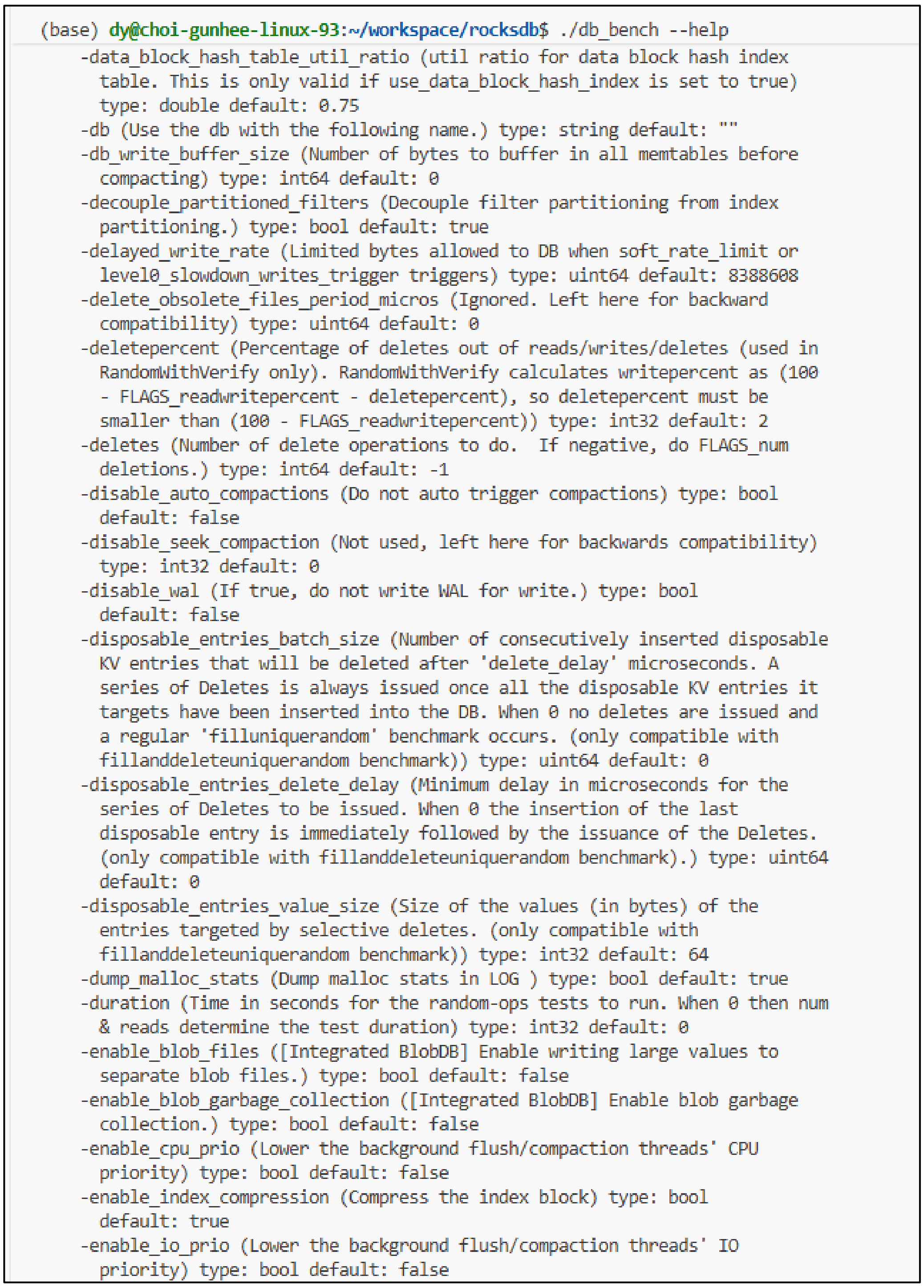


```
** DB Stats **
Uptime(secs): 3.8 total, 3.8 interval
Cumulative writes: 1000K writes, 1000K keys, 1000K commit groups, 1.0 writes per commit group, ingest: 0.12 GB, 33.18 MB/s
Cumulative WAL: 1000K writes, 0 syncs, 1000000.00 writes per sync, written: 0.12 GB, 33.18 MB/s
Cumulative stall: 00:00:0.000 H:M:S, 0.0 percent
Interval writes: 1000K writes, 1000K keys, 1000K commit groups, 1.0 writes per commit group, ingest: 124.93 MB, 33.18 MB/s
Interval WAL: 1000K writes, 0 syncs, 1000000.00 writes per sync, written: 0.12 GB, 33.18 MB/s
Interval stall: 00:00:0.000 H:M:S, 0.0 percent
Write Stall (count): write-buffer-manager-limit-stops: 0

STATISTICS:
rocksdb.block.cache.miss COUNT : 0
rocksdb.block.cache.hit COUNT : 0
rocksdb.block.cache.add COUNT : 0
rocksdb.block.cache.add.failures COUNT : 0
rocksdb.block.cache.index.miss COUNT : 0
rocksdb.block.cache.index.hit COUNT : 0
rocksdb.block.cache.index.add COUNT : 0
rocksdb.block.cache.index.bytes.insert COUNT : 0
rocksdb.block.cache.filter.miss COUNT : 0
rocksdb.block.cache.filter.hit COUNT : 0
rocksdb.block.cache.filter.add COUNT : 0
rocksdb.block.cache.filter.bytes.insert COUNT : 0
rocksdb.block.cache.data.miss COUNT : 0
rocksdb.block.cache.data.hit COUNT : 0
rocksdb.block.cache.data.add COUNT : 0
rocksdb.block.cache.data.bytes.insert COUNT : 0
rocksdb.block.cache.bytes.read COUNT : 0
rocksdb.block.cache.bytes.write COUNT : 0
rocksdb.block.cache.compression.dict.miss COUNT : 0
rocksdb.block.cache.compression.dict.hit COUNT : 0
rocksdb.block.cache.compression.dict.add COUNT : 0
rocksdb.block.cache.compression.dict.bytes.insert COUNT : 0
rocksdb.block.cache.add.redundant COUNT : 0
rocksdb.block.cache.index.add.redundant COUNT : 0
rocksdb.block.cache.filter.add.redundant COUNT : 0
rocksdb.block.cache.data.add.redundant COUNT : 0
rocksdb.block.cache.compression.dict.add.redundant COUNT : 0
rocksdb.secondary.cache.hits COUNT : 0
rocksdb.secondary.cache.filter.hits COUNT : 0
rocksdb.secondary.cache.index.hits COUNT : 0
rocksdb.secondary.cache.data.hits COUNT : 0
rocksdb.compressed.secondary.cache.dummy.hits COUNT : 0
rocksdb.compressed.secondary.cache.hits COUNT : 0
rocksdb.compressed.secondary.cache.promotions COUNT : 0
rocksdb.compressed.secondary.cache.promotion.skips COUNT : 0
```

Benchmarks=stats와 statistics 옵션을 줌으로써 더 자세한 결과를 확인할 수 있음

또한 터미널에서 ./db_bench --help 를 입력하면 db_bench 에서 사용할 수 있는 여러 다양한 옵션들을 출력해줌

따로 옵션들에 대해 설정하지 않은 값은 default: xx 로 지정되어 있어서 자동으로 default 값 사용



```
(base) dy@choi-gunhee-linux-93:~/workspace/rocksdb$ ./db_bench --help
    -data_block_hash_table_util_ratio (util ratio for data block hash index
      table. This is only valid if use_data_block_hash_index is set to true)
      type: double default: 0.75
    -db (Use the db with the following name.) type: string default: ""
    -db_write_buffer_size (Number of bytes to buffer in all memtables before
      compacting) type: int64 default: 0
    -decouple_partitioned_filters (Decouple filter partitioning from index
      partitioning.) type: bool default: true
    -delayed_write_rate (Limited bytes allowed to DB when soft_rate_limit or
      level0_slowdown_writes_trigger triggers) type: uint64 default: 8388608
    -delete_obsolete_files_period_micros (Ignored. Left here for backward
      compatibility) type: uint64 default: 0
    -deletepercent (Percentage of deletes out of reads/writes/deletes (used in
      RandomWithVerify only). RandomWithVerify calculates writepercent as (100
      - FLAGS_readwritepercent - deletepercent), so deletepercent must be
      smaller than (100 - FLAGS_readwritepercent)) type: int32 default: 2
    -deletes (Number of delete operations to do.  If negative, do FLAGS_num
      deletions.) type: int64 default: -1
    -disable_auto_compactions (Do not auto trigger compactions) type: bool
      default: false
    -disable_seek_compaction (Not used, left here for backwards compatibility)
      type: int32 default: 0
    -disable_wal (If true, do not write WAL for write.) type: bool
      default: false
    -disposable_entries_batch_size (Number of consecutively inserted disposable
      KV entries that will be deleted after 'delete_delay' microseconds. A
      series of Deletes is always issued once all the disposable KV entries it
      targets have been inserted into the DB. When 0 no deletes are issued and
      a regular 'filluniquerandom' benchmark occurs. (only compatible with
      fillanddeleteuniquerandom benchmark)) type: uint64 default: 0
    -disposable_entries_delete_delay (Minimum delay in microseconds for the
      series of Deletes to be issued. When 0 the insertion of the last
      disposable entry is immediately followed by the issuance of the Deletes.
      (only compatible with fillanddeleteuniquerandom benchmark).) type: uint64
      default: 0
    -disposable_entries_value_size (Size of the values (in bytes) of the
      entries targeted by selective deletes. (only compatible with
      fillanddeleteuniquerandom benchmark)) type: int32 default: 64
    -dump_malloc_stats (Dump malloc stats in LOG ) type: bool default: true
    -duration (Time in seconds for the random-ops tests to run. When 0 then num
      & reads determine the test duration) type: int32 default: 0
    -enable_blob_files ([Integrated BlobDB] Enable writing large values to
      separate blob files.) type: bool default: false
    -enable_blob_garbage_collection ([Integrated BlobDB] Enable blob garbage
      collection.) type: bool default: false
    -enable_cpu_prio (Lower the background flush/compaction threads' CPU
      priority) type: bool default: false
    -enable_index_compression (Compress the index block) type: bool
      default: true
    -enable_io_prio (Lower the background flush/compaction threads' IO
      priority) type: bool default: false
```

캡쳐된 부분 이외에도 100개가 넘는 옵션들이 존재함

4. 진행 방법

4.1 코드 분석 및 수정

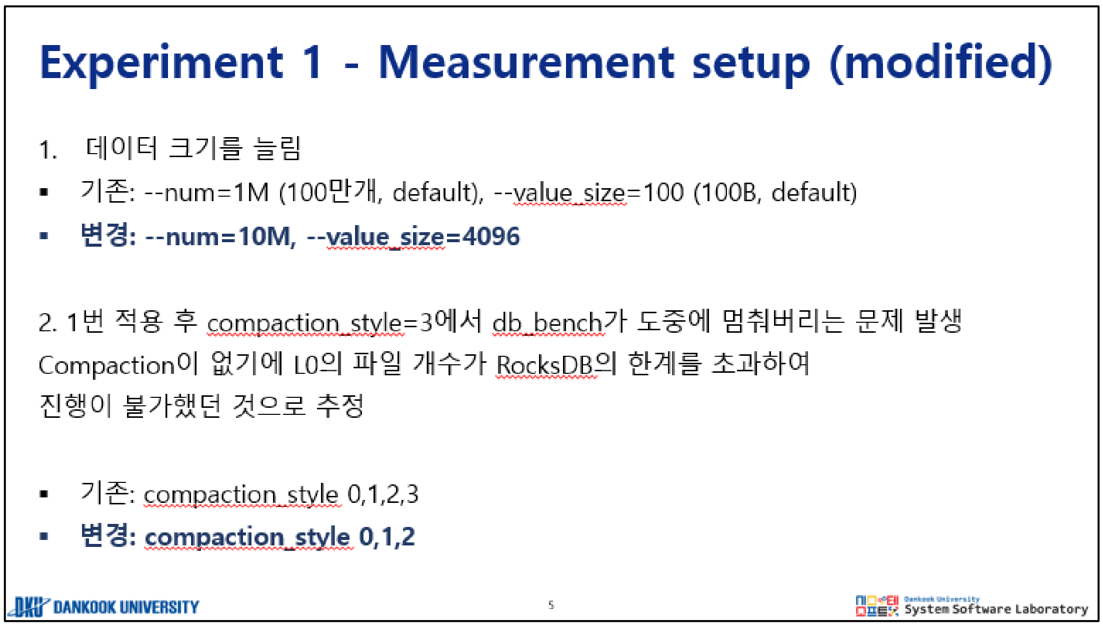
https://github.com/DKU-StarLab/2026_RocksDB_Study/tree/main/presentation_file

깃허브에 업로드된 W0_DayeonWee _How_To_Analyze_RocksDB 자료를 참고

또한 llm에게 물어보는 것도 방법이긴 하나, Hallucination이 있으니 너무 믿진 말 것

4.2 옵션 튜닝

사진에 나와있는 것처럼 옵션을 변경할 수 있음



Experiment 1 - Measurement setup (modified)

1. 데이터 크기를 늘림
- 기존: --num=1M (100만개, default), --value_size=100 (100B, default)
- **변경: --num=10M, --value_size=4096**

2. 1번 적용 후 compaction_style=3에서 db_bench가 도중에 멈춰버리는 문제 발생
Compaction이 없기에 L0의 파일 개수가 RocksDB의 한계를 초과하여
진행이 불가했던 것으로 추정

- 기존: compaction_style 0,1,2,3
- **변경: compaction_style 0,1,2**

DANKOOK UNIVERSITY 5 Dankook University System Software Laboratory

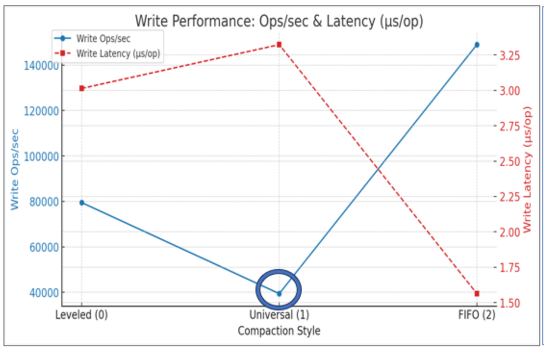
출처: https://github.com/DKU-StarLab/1DanRock/blob/main/presentation_file/Team2/Team2_Week5.pptx

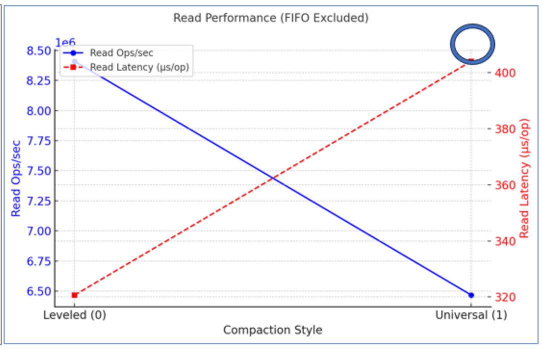
실험을 진행하기 전, 가설을 필수적으로 세워야 함

이 실험을 진행하면 어떠한 이유 때문에 이렇게 결과가 나올 것이다! 와 같은 가설을 세워야 함

또한 이 실험을 왜 진행했는지 motivation이 있으면 좋음

Experiment 1 – Result (graph)





DKU DANKOOK UNIVERSITY 8 Dankook University System Software Laboratory

다음과 같이 실험 결과를 그래프로 뽑아볼 수 있음

그래프는 jupyter notebook (python)이나 엑셀을 통해서 그래프를 그려볼 수 있음

이를 통해 결과를 분석해볼 수 있음

화이팅..!