harperdb.io
hello@harperdb.io
2420 17th St. Suite 270, Denver, CO. 80202

HarperDB

BENCHMARK REPORT | December 19th, 2022

Lookup Speed Comparison of Normalized and Denormalized Data Models at Scale using HarperDB

Outcome

This comparison demonstrates that HarperDB enables normalized data to be a practical storage solution with minimal impact on read performance. In this test, denormalization provides a negligible advantage of less than two milliseconds over normalized data for read operations but causes total space on disk, write times, and insert times to be dramatically reduced.

harperdb.io
hello@harperdb.io
2420 17th St. Suite 270, Denver, CO. 80202

HarperDB

Data Model Overview

Normalized Data

Normalized data is organized into tables, each representing a specific type of information. This structure divides data into logical groups to minimize redundancy and ensure consistency. Relationships between tables are established through keys, allowing for efficient data retrieval and updating. Normalized data models help ensure data integrity and reduce the risk of errors or inconsistencies in the data.

Denormalized Data

Denormalized data combines multiple types of data into a single table. This can result in data duplication and a larger table size. Accordingly, it consumes more of a database's in-memory cache. It can also increase the risk of inconsistencies in the data and make updates more resource-intensive since data needs to be updated in multiple locations. Historically, the upside to denormalization is lookup speed because the system just needs to perform one look-up rather than multiple.

Summary of Results

- Normalized data's performance scales significantly better.
- Denormalized data's speed advantage drops off when the size on disk exceeds system RAM.
- On average, normalized data has 12.12 times more throughput on write than denormalized.
- On average, denormalized data is 13.1 times larger than normalized data on disk.

Performance was compared for normalized and denormalized data models. Multiple tests were run against datasets of increasing size and with increasing user concurrency on a single server.

harperdb.io
hello@harperdb.io
2420 17th St. Suite 270, Denver, CO. 80202

HarperDB

Datasets were intended to simulate various user counts, starting with 20,000 users and increasing by an order of magnitude for each test, with a final simulation of 200,000,000 users. Each user has 150 entitlements that represent purchases of various digital items. Entitlements were stored as part of the user's record when testing the denormalized data model and as a separate table while testing normalized data.

Entitlements were randomly selected from a pool of entitlements that varied in size based on the user base. The number of entitlements was 10% of the number of users. For example, when testing 20 million users, a list of 2 million potential entitlements was created, then each user was associated with 150 of those entitlements.

For a smaller dataset representing 2 million users, the read throughput for the denormalized model was 30K/s with sub-millisecond response times. In contrast, the normalized model delivered a throughput of 12K/s with response times under 2ms. However, once the database size exceeds the server's available RAM, the advantage of a normalized data structure becomes self-evident.

For 20 million users, the size of the denormalized data exceeded the available RAM. This resulted in a throughput drop to 9K/s and 3-4ms response times. The normalized data's throughput—being far smaller on disk—remained steady at 12K/s, with response times increasing only slightly to just over 2ms.

For 200 million users, the size of the normalized data also exceeded the available RAM. Throughput dropped only slightly to 9K/s, with median response times increasing only slightly to 2.6ms.

We could not test 200 million denormalized records, as loading that much data on disk would have taken far too long. Another advantage of the normalized data structure is writing performance, with write throughput remaining consistently 12 times higher than for denormalized (13,000 records/second vs 1,100 records/second).

harperdb.io
hello@harperdb.io
2420 17th St. Suite 270, Denver, CO. 80202

Methodology and Hardware Specs

Benchmark testing was performed against HarperDB 4.0.0 running on a Linode VM in the Atlanta, GA region.

Hardware:
50 CPU Cores - AMD EPYC 7642
128GB RAM
2,500 GB Storage, ext4
Ubuntu 22.04

Tests were executed against User record sets of 20K, 200K, 2MM, 20MM, and 200MM, where each User has 150 associated entitlements. For each increment of record sets, we ran tests to show performance as we increased the number of concurrent connections (cc) which scaled as follows: 1cc, 50cc, 500cc, 1000cc, and 5000cc. The goal of scaling concurrent connections was to demonstrate not just optimistic performance with a single API call, but to show how HarperDB performs as we scale concurrent requests against both the normalized and denormalized data sets.

For each increment in the size of data sets, new data was loaded into HarperDB tables. Data was loaded with sequential primary key values to optimize data ingest performance.

Every test iteration was executed to compare the performance of normalized data sets versus denormalized data sets. The load was applied to HarperDB via another dedicated server, with equal specs, in the Atlanta, GA region running k6, an open-source, high-scale load testing tool. HarperDB was running with the HTTP protocol, and we were executing the requests with Keep-Alive enabled. Tests were executed for 300 seconds, where we recorded summary statistics for each test run and data for every request performed. All times reported in this document are the time it took HarperDB to process requests, recorded via the HarperDB server-timing header, and not the total response time of a request, which would include variable Internet latency.

harperdb.io
hello@harperdb.io
2420 17th St. Suite 270, Denver, CO. 80202

HarperDB Storage Technology

HarperDB's storage engine is based on advanced memory mapping technology that enables extremely fast and efficient data access with minimal overhead from system calls, avoiding redundant caching structures and fully leveraging operating system caching capabilities and machine resources. HarperDB's storage engine (LMDB) uses a highly optimized B-tree structure for fast O(log n) data access. This is combined with an extremely tight in-process path from storage to application code (Custom Functions), which enables a high level of flexibility in aggregating data while achieving high-performance results.

By using memory maps, HarperDB directly uses the operating system disk cache as the database cache. This allows HarperDB processes to maintain consistent in-memory processing performance while dynamically leveraging the rest of available RAM for database caching to maximize how much data can be effectively cached. This provides efficient memory usage and fast responses for data that is accessed multiple times.

However, as a database size grows beyond the available RAM, there is an inherent need for some database lookups to involve disk reads to fetch data. Precisely estimating the percentage of operations that will require disk reads can be challenging and involves careful analysis of database size and probability distributions of different requests, but generally speaking, as the database grows beyond the size of RAM and to the degree that accessed entitlement are widely distributed, more disk reads will be required.

Normalization Strategy

Traditional relational databases generally have highly normalized data structures and relationships. This can yield very compact and efficient data structures from a storage perspective, accordingly making updates very fast. But, the deep relations and necessary joins are often onerous and preclude high-performance, low-latency access requirements. On the other hand, caching servers often use highly denormalized data, which requires fewer lookups. However, simplistic caching servers can require extreme levels of denormalization that yield very inefficient storage, resulting in poor caching characteristics (since databases can grow well beyond RAM), and extremely slow and cumbersome updates.

harperdb.io
hello@harperdb.io
2420 17th St. Suite 270, Denver, CO. 80202

In this prototype, we are pursuing the strategy of a “balanced” normalization. There are often natural relations in the data that can be flattened and denormalized. Selective denormalization of data that is always user or entitlement-specific eliminates deeper levels of relations and joins for fast access. However, we believe there is also a significant benefit in maintaining the normalization of entitlement as a separate table from users since user accounts often reference hundreds of entitlements. By maintaining this normalization, storing users and records can be very efficient. Consequently, entitlements that correspond to multiple users can quickly and easily be updated, storage requirements are dramatically reduced, and with the large scale of data required, a much higher percentage of data can be maintained in memory caches for faster access. With HarperDB’s flexible data schemas and model, in combination with the ability to write customized aggregation queries to join data in Custom Functions, HarperDB facilitates this “balanced” normalization approach.

To test this prototype, we have benchmarked normalization (of users and entitlement as separate tables) versus a full denormalized data model with increasing database size. Before benchmarking, our hypothesis was that a denormalized data model would be fastest with smaller databases that fit in memory since each request essentially involves a single B-tree lookup. But, as the database grows in size, we expect the normalized database to be much more compact. As the database exceeds the size of available RAM, we expect to maintain a higher percentage of cache hits and, therefore, a better scaling of performance with the normalized data model. The more efficient data storage model of the normalized data will result in much faster write/update performance and reduced storage requirements and cost.

harperdb.io
hello@harperdb.io
2420 17th St. Suite 270, Denver, CO. 80202

HarperDB

Results

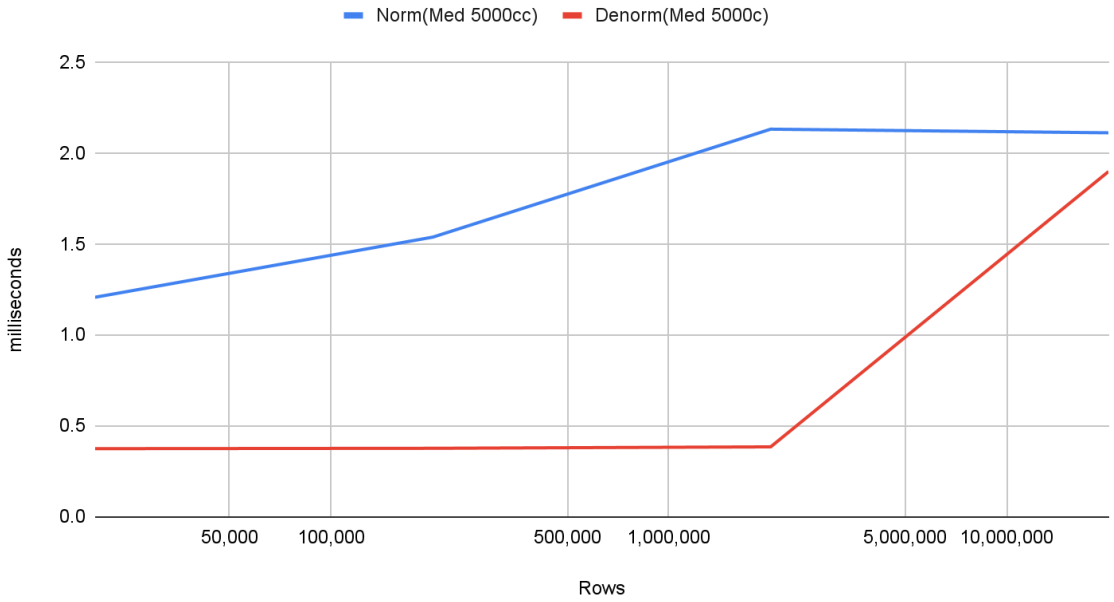
For smaller databases (2 million users and fewer), benchmarks of denormalized and normalized both yield extremely fast response times, but as expected, the denormalized model is fastest with nearly instantaneous, sub-millisecond responses. The throughput of the denormalized model (on a single server) is roughly 30K/s, whereas the normalized model is about 12K/s.

However, as the database grows to larger sizes (20 million users), the benchmarks distinctly shift, as hypothesized. At this size, the denormalized database is about twice the size of available RAM, and a significant percentage of the requests require disk reads, dropping throughput down to about 9K/s with 3-4 millisecond access time under load. On the other hand, the normalized database is significantly smaller than the denormalized database and continues to easily fit in RAM. This is demonstrated as the throughput is almost unchanged at this scale, still maintaining 12K/s and 2-3 millisecond access times.

In the full production scenario, we would anticipate that even the normalized database may exceed available RAM, but with realistic distributions of data accesses, the vast majority of requests could be accessible through in-memory cached data. With full large-scale handling of the entire user database, this normalized data model approach appears to strike a balanced approach to fast data access with minimal data fetches while maintaining the necessary normalization for efficient storage and updates.

harperdb.io
hello@harperdb.io
2420 17th St. Suite 270, Denver, CO. 80202

HarperDB



harperdb.io
hello@harperdb.io
2420 17th St. Suite 270, Denver, CO. 80202

HarperDB

Performance Metrics

Write Performance and Size on Disk

- On average, normalized data has 12.12 times more throughput on write than denormalized.
- On average, denormalized data is 13.1 times larger than normalized data on disk.

	Denormalized			Normalized		
User Records	**Size on Disk(MB)**	**Time to Create**	**Insert Rate**	**Size on Disk(MB)**	**Time to Create**	**Insert Rate**
20,000	237	18.5	1,189	14	1.5	13333
200,000	2400	191	1,152	138	17	11765
2,000,000	24000	1911	1,151	2600	151	13245
20,000,000	237000	19670	1,118	26000	1602	12484
200,000,000	N/A	N/A	N/A	219,000	21224	9423

We project that: 200MM rows would take 50 hours to insert denormalized and would take up 2.5TB on disk.

As can be seen from the above table, loading denormalized data takes significantly longer than normalized. Another drawback to a denormalized data approach is the footprint on disk is significantly higher. Maintaining a normalized data set will be significantly more performant as only the referenced entitlement would need to be updated instead of a denormalized data set where all users would need to be modified, requiring many more writes to the system.

harperdb.io
hello@harperdb.io
2420 17th St. Suite 270, Denver, CO. 80202

HarperDB

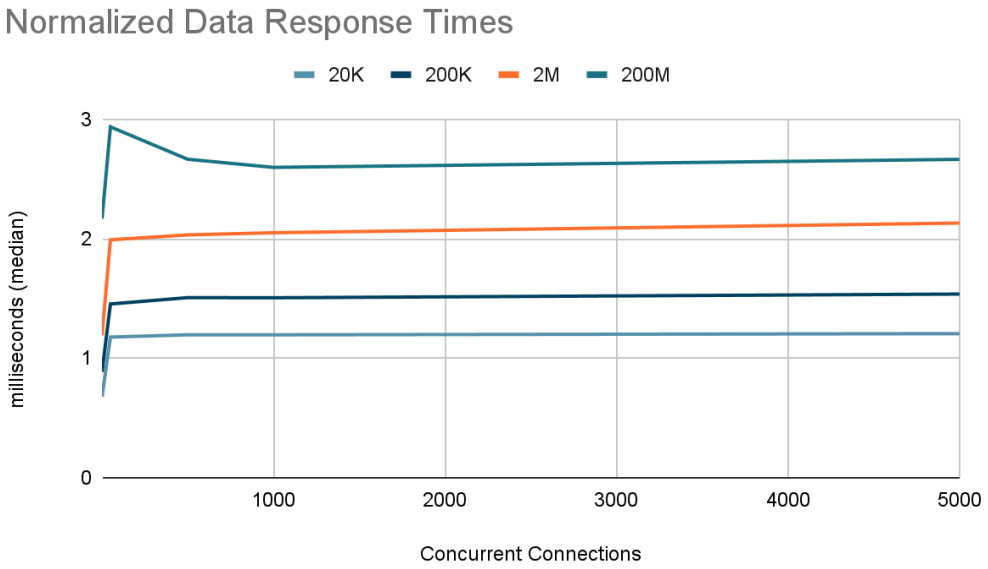
Read Performance

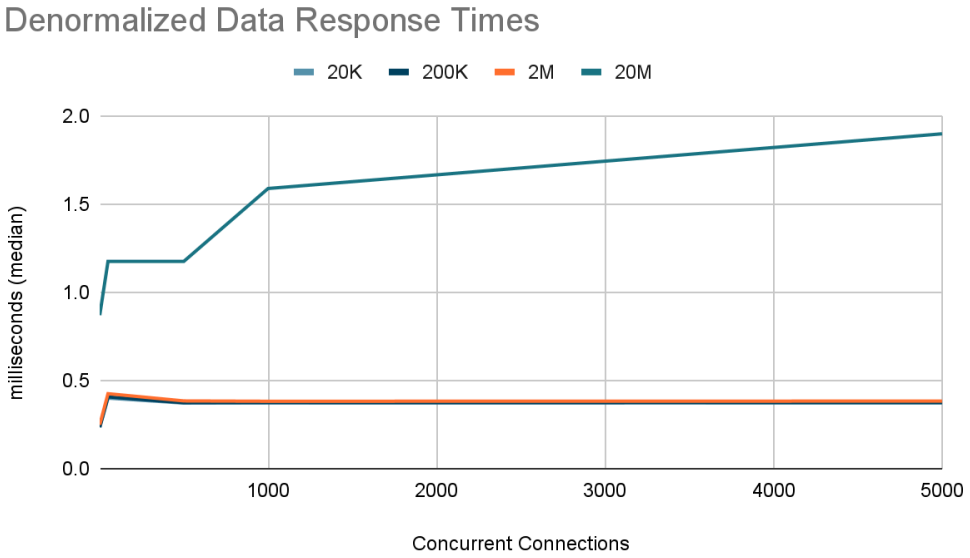
	Denormalized		Normalized	
Users Records	Median Response (ms)	Reads (s)	Median Response (ms)	Reads (s)
20,000	0.373	40K	1.206	26K
200,000	0.375	39K	1.537	22K
2,000,000	0.383	36K	2.131	12K
20,000,000	1.898	8K	2.111	12K
200,000,000	N/A	N/A	2.664	9K

harperdb.io
hello@harperdb.io
2420 17th St. Suite 270, Denver, CO. 80202

HarperDB

Graphs





harperdb.io
hello@harperdb.io
2420 17th St. Suite 270, Denver, CO. 80202

HarperDB

Detailed Read Results

20 Thousand Rows										
	Normalized					Denormalized				
Concurrent Connections	1	50	500	1,000	5,000	1	50	500	1,000	5,000
Total Reads	128,687	4,851,881	9,589,017	9,290,853	7,857,566	162,502	6,291,355	12,714,593	12,569,039	11,999,688
Data Received	5,002,577,355	611,614,860	762,606,402	171,787,921	454,435,108	644,848,936	327,815,030	425,239,894	264,963,542	990,950,719
Response Time (ms)										
min	0.596	0.637	0.63	0.622	0.612	0.159	0.161	0.156	0.155	0.156
med	0.678	1.177	1.196	1.196	1.206	0.235	0.4	0.372	0.374	0.373
max	7.443	28.737	104.442	131.388	196.165	4.145	150.833	164.645	226.246	169.669
avg	0.689	1.77	1.212	1.211	1.394	0.248	0.415	0.489	0.498	0.537
p90	0.734	1.449	1.303	1.304	1.381	0.274	0.552	0.472	0.474	0.498
p95	0.769	1.532	1.342	1.343	1.485	0.306	0.596	0.545	0.54	0.575
Total Time (seconds)	300	300	300	300	300	300	300	300	300	300
Reads/Sec	429	16,173	31,963	30,970	26,192	542	20,971	42,382	41,897	39,999

200 Thousand Rows										
	Normalized					Denormalized				
Concurrent Connections	1	50	500	1,000	5,000	1	50	500	1,000	5,000
Total Reads	116,041	4,447,146	7,948,291	7,846,991	6,642,713	161,599	6,202,211	12,348,147	12,146,702	11,579,651
Data Received	545,590,619	207,784,826	357,787,012	389,548,061	214,462,507	643,971,472	235,184,478	353,902,720	064,600,353	768,932,333

harperdb.io
hello@harperdb.io
2420 17th St. Suite 270, Denver, CO. 80202

HarperDB

Response Time (ms)										
min	0.719	0.774	0.84	0.738	0.844	0.163	0.166	0.156	0.159	0.16
med	0.887	1.454	1.507	1.506	1.537	0.235	0.407	0.374	0.374	0.375
max	12.48	133.103	96.795	117.962	168.881	2.515	130.203	209.792	159.492	170.252
avg	0.91	1.817	1.542	1.543	1.772	0.249	0.425	0.5	0.5	0.527
p90	0.956	1.736	1.607	1.61	1.713	0.27	0.56	0.478	0.474	0.503
p95	1.023	1.817	1.649	1.653	1.85	0.302	0.606	0.561	0.543	0.582
Total Time (seconds)	300	300	300	300	300	300	300	300	300	300
Requests per Second	387	14,824	26,494	26,157	22,142	539	20,674	38,599	40,489	38,599

2 Million Rows										
	Normalized					Denormalized				
Concurrent Connections	1	50	500	1,000	5,000	1	50	500	1,000	5,000
Total Reads	98,547	3,193,986	4,274,124	4,239,849	3,828,260	159,508	6,221,383	10,331,057	10,744,152	10,788,058
Data Received	390,101,031	084,557,709	724,154,501	370,240,588	123,364,693	607,747,241	718,722,888	436,216,914	369,632,789	637,432,395
Response Time (ms)										
min	0.893	1.023	1.013	1.01	0.999	0.166	0.172	0.169	0.166	0.169
med	1.191	1.991	2.032	2.05	2.131	0.249	0.425	0.384	0.382	0.383
max	11.003	107.982	348.558	315.665	249.709	7.162	247.173	158.944	160.977	189.047
avg	1.247	2.072	2.729	2.755	3.067	0.263	0.445	0.501	0.499	0.525
p90	1.304	2.347	2.287	2.315	2.508	0.29	0.584	0.492	0.49	0.516
p95	1.456	2.784	4.102	4.073	7.157	0.33	0.632	0.576	0.565	0.596
Total Time (seconds)	300	300	300	300	300	300	300	300	300	300

harperdb.io
hello@harperdb.io
2420 17th St. Suite 270, Denver, CO. 80202

HarperDB

Requests per Second	328	10,647	14,247	14,133	12,761	532	20,738	34,437	35,814	35,960

20 Million Rows										
	Normalized					Denormalized				
Concurrent Connections	1	50	500	1,000	5,000	1	50	500	1,000	5,000
Total Reads	96,315	3,111,867	4,032,780	3,993,674	3,674,415	117,392	2,989,394	3,340,637	3,055,762	2,250,953
Data Received	331,134,592	780,147,963	411,512,616	355,261,908	156,092,353	408,820,324	306,054,719	,769,573.00	,733,594,570	363,706,196
Response Time (ms)										
min	1.07	1.006	1.162	1.141	1.047	0.17	0.191	0.178	0.175	0.178
med	1.216	1.984	2.036	2.056	2.111	0.87	1.175	1.175	1.588	1.898
max	11.339	187.502	220.732	257.491	229.571	23.626	2646.727	1863.303	2948.391	2704.043
avg	1.282	2.084	2.772	2.833	3.083	0.884	2.692	3.503	3.836	4.886
p90	1.349	2.363	2.328	2.36	2.505	1.451	3.22	3.961	4.016	6.51
p95	1.546	3.013	4.105	4.405	7.081	1.586	4.865	5.889	6.474	10.783
Total Time (seconds)	300	300	300	300	300	300	300	300	300	300
Requests per Second	321	10,373	13,443	13,312	12,248	391	9,965	7,503	10,186	7,503

200 Million Rows										
	Normalized					Denormalized				
Concurrent Connections	1	50	500	1,000	5,000	1	50	500	1,000	5,000
Total Reads	72,732	1,323,121	2,314,850	2,575,355	2,481,470					

harperdb.io
hello@harperdb.io
2420 17th St. Suite 270, Denver, CO. 80202

HarperDB

Data Received	914,928,265	027,972,875	774,581,431	214,711,359	451,905,254					
Response Time (ms)										
min	1.116	1.204	1.054	1.183	1.014					
med	2.167	2.936	2.665	2.597	2.664					
max	98.556	5928.125	1734.754	1007.564	770.276					
avg	2.26	6.703	4.389	3.844	4.107					
p90	3.103	11.768	6.883	5.556	6.341					
p95	3.695	19.698	11.341	8.81	10.544					
Total Time (seconds)	300	300	300	300	300	300	300	300	300	300
Requests per Second	242	4,410	7,716	8,585	8,272	0	0	0	0	0

harperdb.io
hello@harperdb.io
2420 17th St. Suite 270, Denver, CO. 80202

HarperDB

Data Models

Table	Data Type
Entitlements	
id	int
activeDate	Date
inactiveDate	Date
useLimit	int
useCount	int
featureType	string(4)
packageType	string(4)
activeFlag	int
Normalized Users	
id	int
entitlements.ids	[int]
Denormalized Users	
id	int
entitlements.data	[Entitlement]

harperdb.io
hello@harperdb.io
2420 17th St. Suite 270, Denver, CO. 80202

HarperDB

Test Data

The raw load test data has been uploaded to Linode and is publically available at http://k6-data.us-southeast-1.linodeobjects.com/harperdb-normalized-denormalized-rawtestdata.tgz

The file is 20GB compressed and 475GB uncompressed, containing CSV files with data for each of the requests and JSON files with summaries.

The filenames are in the format of STRUCTURE-ROWS-CC.

harperdb.io
hello@harperdb.io
2420 17th St. Suite 270, Denver, CO. 80202

HarperDB

Custom Functions

Two Custom Function API endpoints were created:

`/denorm/:userId`
This API fetches the denormalized User based on the userId.

`/norm/:userId`
This API fetches the normalized User based on the userId. Further fetches are run against the Entitlements table based on the User.entitlements.ids array. The Entitlements are added to the User object and returned in the response.