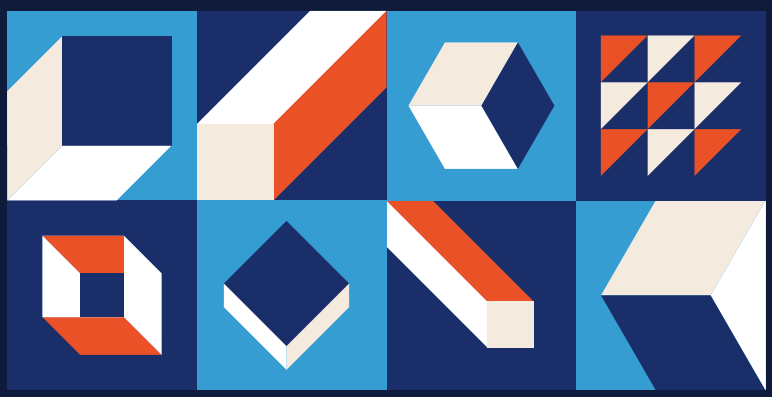
PingCAP TiDB

Can There Be One Database to Rule Them All?

Enabling Data Agility with an HTAP Database



www.pingcap.com

Table of Contents

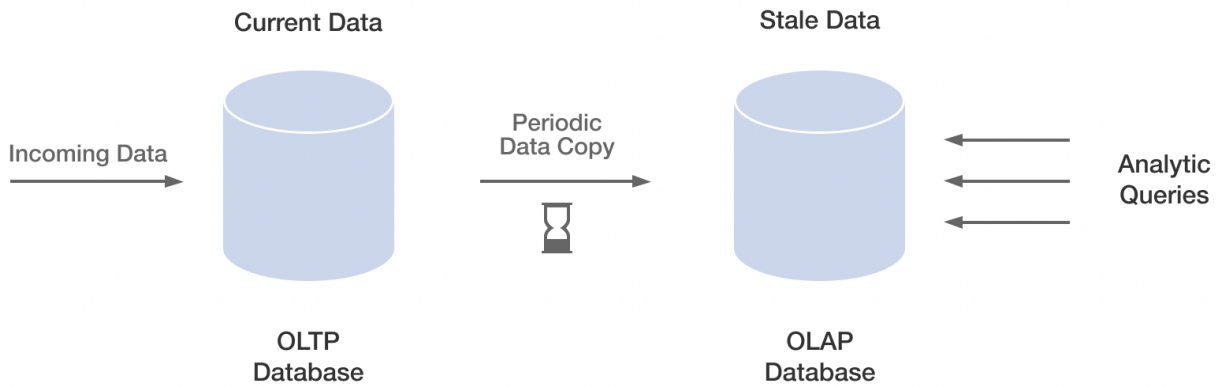
Overview

Database users have always been required to make choices regarding how they store and access their data. Do we use defined schemas, or do we go with a free-form document database? Do we want MySQL or PostgreSQL? Open or closed source?

These questions and more have been faced and answered by many. But one issue has loomed large over the database world. How do I get the fast transaction processing to support my user activity while providing analytical access to that data to those who need it?

The multiple database solution

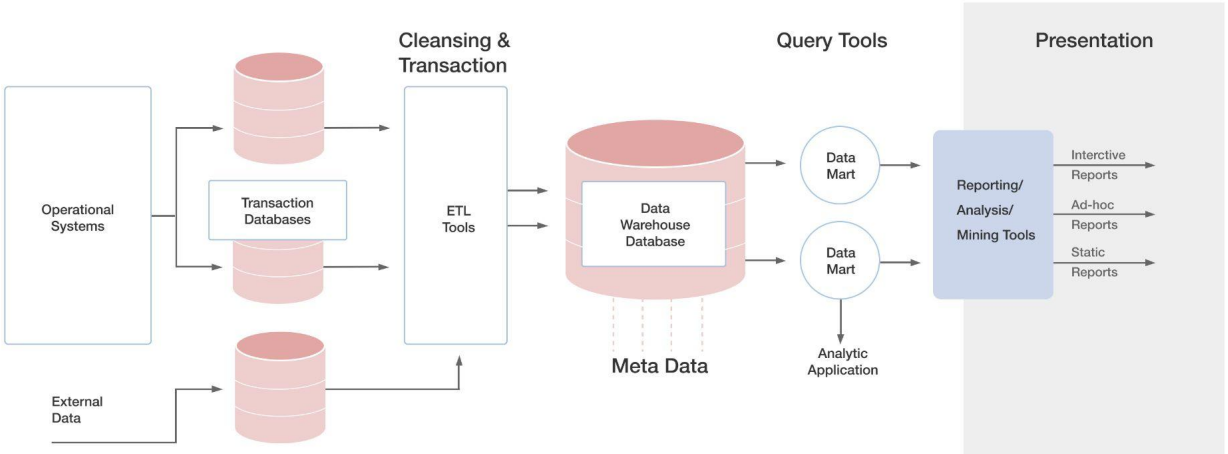
In many cases, organizations used two different databases: one that is optimized for transactions and one that is optimized for analytics. The biggest problem here is the need to somehow copy data from the transactional database to the analytical database. Many companies offered solutions to that issue, but few looked at resolving the underlying problem, which is that data copying inherently introduces data latency. Simply put, if a transaction is processed at a timestamp of 10:16:32, that record is not immediately available in the analytical system until the copy activity completes. You can run the copy activity frequently, but this is an inefficient use of resources and cost.



Multiple database solution

Additionally, in an environment like this, you need to maintain two distinct databases, adding to the cost and increasing complexity. You are also unlikely to meet the demands of an increasingly time-sensitive user community. Imagine if you made a bank transaction, but the record of that transaction sat in the transactional database for some amount of time before it could be reported in your banking app. Whether that delay is one day, 12 hours, one hour, or less, it is still a delay and will result in user frustration.

In a banking scenario, this delay can also enable fraudulent transactions to be processed. Without the ability to update data quickly, it is possible for someone to create multiple similar transactions, perhaps transferring and withdrawing money, without the bank being able to quickly and accurately identify the transactions as fraud.



Complexity in a multiple database solution

What is needed is a database that can act as a single source of truth, providing you with real-time analytics in your database of record. TiDB writes data to both a transactional and analytical store at the time of the data commit, so it provides such a capability. The transactional data is written to TiKV, our row store, while the analytical data is written to TiFlash, which is a column store. Replication takes place in real-time, ensuring that the data reported by an analytical query is fully up to date.

The single source of truth

There are many benefits to a Hybrid Transactional and Analytical Processing (HTAP) database like TiDB beyond its real-time reporting capabilities. These include

- Reduced operating costs
- Lower probability of errors
- Faster development cycles
- Centralized compliance control
- Enhanced time to market with new features

There is less overhead, both hardware and labor, since you are supporting a single database instead of two (or more) databases. There is a reduced likelihood of error since you no longer need to rely on an ETL process to copy data between the databases. Best of all, you benefit from a drastic reduction in development time since all the data is written to, stored in, and queried from the same source.

Since all of the data is stored in a single database, meeting compliance requirements is easier. Now, auditors have a single location to manage all regulatory conditions. With fewer databases and people involved, iterative development is enabled. Your development team can make changes and see the impact quickly without the need to involve other teams. This is true for both new projects and updates to existing projects.

In addition to being a single source of truth, the One Database to Rule Them All needs to scale easily to accommodate change, support MySQL, and be highly secure. Even better would be the ability to scale compute separately from storage (and vice versa) and provide a distributed solution that is fully cloud-enabled.

A sample case

GitHub provides internet hosting for software development and is used to host most open-source projects. As of June 2022, GitHub reports having over 83 million developers and more than 200 million repositories. This is obviously a massive data

source and one that is constantly changing, as users access data and make changes to the information in the repositories.

With this sort of data capacity, performing any viable analysis is a daunting task. You have high traffic volume, plus a massive amount of historical data. Trying to analyze this data becomes a challenging task, particularly since it is the most recent data that is likely to be important. Knowing what repository was most popular last month is helpful, but knowing which repository is seeing the most current activity provides you with actionable data.

What we used to do

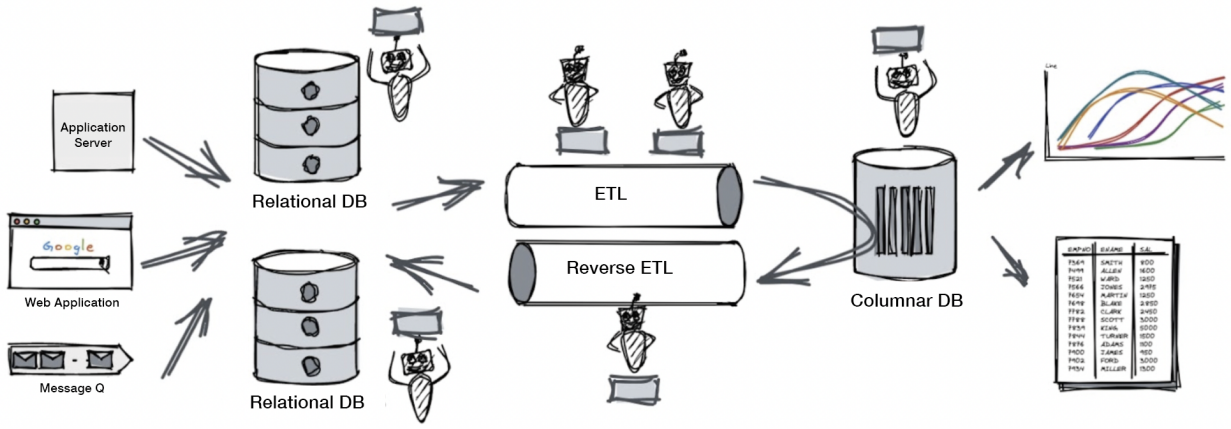
Looking at a legacy type of solution to this, it is likely that one database would be used to process the transactions and a second database would be used to handle the analytics. Unfortunately, there are multiple problems with such a solution.

First, on the transactional side, we would need a robust architecture that could support the current and expected future traffic. A solution like GitHub could have usage spikes for any number of reasons, ranging from the release of a new version in a popular repository to the unexpected consequence of a weather event causing large numbers of people to work from home. One of these events can be planned around, but the weather event, and others like it, is more variable in duration and impact.

Second, on the analytical side, we need another robust architecture plus a plan for scaling as needed. Given the number of repositories and high usage traffic, this data environment will grow to a huge size and will continue to grow over time. The maintenance of an environment to support this is challenging.

Last, in this legacy solution, there is the need for some way to copy data from the transactional side to the analytical side. This process inherently introduces data lag, where information exists in the transactional environment but has not yet been copied to the analytical side. The other issue is that you need to support, monitor, and manage the copy activity to ensure that it is completing as expected. Of course, both database environments also need to be monitored, supported, and managed as well.

These issues create a need for large teams of people to handle the products and the creation and maintenance of multiple hardy database environments. This leads to increased costs, with the added burden of additional points of failure.



Resource requirements for a multiple database solution

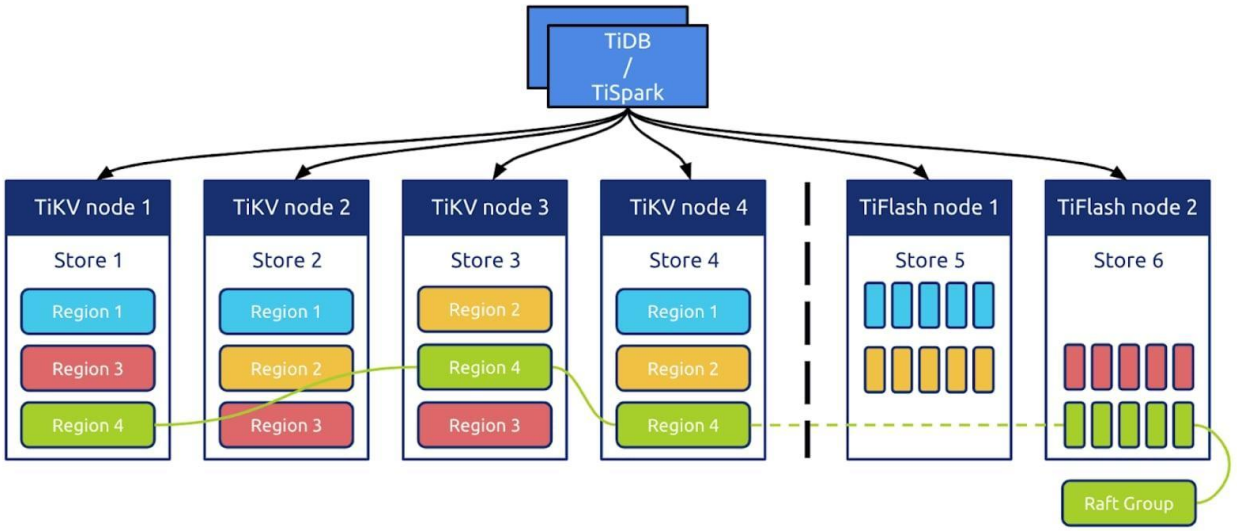
What is possible

A single database that could handle both your transaction load and provide a quick response to analytic queries would be ideal. Such a database would need to have a way to handle a high volume of transactions, manage replication to ensure data consistency, and deliver speedy responses to analytical queries quickly and accurately.

TiDB delivers on that promise. When a write request is made to TiDB, the row store, called TiKV, quickly processes that request. Confirmation of the write is dependent on the record being recorded in a plurality of the nodes. Replication is by nature managed by TiDB via the Raft protocol.

In addition to replicating the record to additional row store nodes, TiDB can also replicate to nodes that are dedicated to storing the data in a column-based format. TiFlash, the column store, is optional but, when enabled, provides an ideal method for responding to analytical queries. Since these queries tend to rely on fewer columns, and often aggregate data across many records, a column-store provides a faster and

more efficient response. This replication is managed via the Raft protocol, providing real-time analytics.



TiDB architecture

The query optimizer in TiDB makes the decision whether to use the row-store, column-store, or a combination of both to respond to each query. This means that your end-users do not need to wonder if they are querying the right database for their specific usage request; instead, the query optimizer makes that decision, ensuring the most efficient use of resources.

```
explain SELECT    country_code,
                  COUNT(DISTINCT actor_id) AS users_count
        FROM github_events
        JOIN db_repos db ON db.id = github_events.repo_id
        JOIN users u     ON u.login = github_events.actor_login
        WHERE event_year = 2021
          AND github_events.type IN ('IssuesEvent', 'PullRequestEvent', 'IssueCommentEvent')
          AND country_code IS NOT NULL
          GROUP BY 1
          ORDER BY 2 DESC
          LIMIT 10;

+-----------------------------------------------------+-----------------+-----------+-----------------------------------------+-----------------------------------------+
| id                                                  | estRows         | task      | access object                           | operator info                           |
+-----------------------------------------------------+-----------------+-----------+-----------------------------------------+-----------------------------------------+
| Projection_19                                       | 10.00           | root      |                                         | gharchive_dev.users.country_
| └─TopN_22                                           | 10.00           | root      |                                         | Column#50:desc, offset:0, co
|   └─HashAgg_27                                      | 20.37           | root      |                                         | group by:gharchive_dev.users
|     └─IndexJoin_38                                  | 853.31          | root      |                                         | inner join, inner:IndexLookU
|       ├─TableReader_69(Build)                       | 838.43          | root      | table of db, partition:pull_request_event, | data:ExchangeSender_68
|       │ └─ExchangeSender_68                        | 838.43          | mpp[tiflash] |                                      | ExchangeType: PassThrough
|       │   └─HashJoin_65                            | 838.43          | mpp[tiflash] |                                      | inner join, equal:[eq(Colum
|       │     ├─ExchangeReceiver_55(Build)           | 46.00           | mpp[tiflash] |                                      |
|       │     │ └─ExchangeSender_54                 | 46.00           | mpp[tiflash] |                                      | ExchangeType: Broadcast
|       │     │   └─Projection_52                   | 46.00           | mpp[tiflash] |                                      | cast(gharchive_dev.db_repos.
|       │     │     └─TableFullScan_53              | 46.00           | mpp[tiflash] | table:db                             | keep order:false
|       │     └─Projection_56(Probe)                | 169296553.95    | mpp[tiflash] |                                      | gharchive_dev.github_events.
|       │       └─Selection_58                      | 169296553.95    | mpp[tiflash] |                                      | eq(gharchive_dev.github_ever
|       │         └─TableFullScan_57                | 4642033314.00   | mpp[tiflash] | table:github_events                  | keep order:false, PartitionT
|       └─IndexLookUp_37(Probe)                      | 1.02            | root      |                                         |
|         ├─IndexRangeScan_34(Build)                 | 12.49           | cop[tikv] | table:u, index:index_login_on_users(login) | range: decided by [eq(gharc
|         └─Selection_36(Probe)                      | 1.02            | cop[tikv] |                                      | not(isnull(gharchive_dev.us
|           └─TableRowIDScan_35                      | 12.49           | cop[tikv] | table:u                              | keep order:false
+-----------------------------------------------------+-----------------+-----------+-----------------------------------------+-----------------------------------------+
```


Query optimizer in TiDB

TiDB was used to build out a demonstration site called ossinsight.io. This site accesses over 4.7 billion events in GitHub and provides a real-time view of activity. Site information is updated in real-time, so, as users take action within GitHub, that activity is reported on the site. The most amazing fact about this site is that it was created in only seven days. The architecture of TiDB enables faster development cycles and leads to faster releases since you are dealing with a single database, rather than multiple, disparate environments.

OSS Insight

The analytic capabilities of TiDB also mean that you are better able to target your products to your user's needs. In OSS Insight, an insight tool of open source projects powered by TiDB, you can view attributes of any of the repositories in GitHub and compare them against each other. In a business setting, this provides real-time results that you can use to best the competition.

Additionally, TiDB enables you to scale your environment to meet your needs. You can scale compute and storage power independently, so you do not need to add more

capabilities than you need. Not only does this deliver an environment that is consistently right-sized, but it also saves you money, since you only need to add the resource you require.

What this means to you

Since a single TiDB cluster provides the speed you need for transaction processing and access to real-time analytics, you are saving on hardware and people costs. Less hardware is needed since you have a single database, rather than multiple databases storing the same information. TiDB eliminates the need for an ETL process that transfers data from the transactional database to the analytical database. Your end-users also benefit since their queries always run in the most efficient way possible.

Since you are now maintaining a single database as your source of truth, you need fewer people to manage and maintain it. Not only does this reduce your personnel costs, but it also enables you to focus your teams on the application rather than the database. Your Dev team can run through sprints more quickly and Sys Ops is freed from the worries and hassle of ensuring schema changes are repeated through multiple environments. Happier teams also reduce turnover and the constant need to retrain new hires.

A single database speeds time to market by, on average, a factor of 3. So, a project that would previously take three months to complete can be finalized in one month or less. This also factors into iterative development cycles, enabling them to complete more quickly.

One Database to Rule Them All also means that there is less need for maintenance. If you live in a multi-car household, you must monitor and maintain each vehicle separately. The same is true in a multi-database world. TiDB provides a single database environment to monitor and maintain, again freeing resources for other uses.

From an operating cost perspective, we can see significant savings with a product like TiDB. If we assume a database that is managing 6T of data, which is significantly less than what is seen in GitHub, running on AWS Aurora and Redshift, we can recognize

massive reductions in operating and capital costs. The old environment requires 10 engineers to manage and maintain it; the new TiDB environment requires only 2 engineers. If we assume the cost per engineer to be $150,000 per year, the old environment had a people cost of $1,500,000 while the new environment has a cost of only $300,000. This is an 80% savings in people cost alone. When we factor in the cost reduction due to a simplified environment, we see an overall operating and capital expense of just under 70% in total. Couple these savings with a 3x faster time to market and you have a winning combination.

These savings could be used in multiple ways. You could reassign the people resources to new projects, some of which can help to grow company revenue. You could also allocate some of the savings to new or improved hardware environments. Alternatively, these savings could simply benefit your bottom line.

Factor these actual cost savings in with a reduction in environmental complexity plus a faster time to market and you have a winning combination. With TiDB, you see cost reduction but also improved productivity and focus, since the daily drudgery of managing databases is reduced. Lower costs, happier and more productive employees lead to greater value from your database.

One database to rule them all?

TiDB can become your database of record with its transaction processing capabilities, while also providing real-time analytics. TiDB delivers

- Transactions at speed and real-time analytics
- Reduced hardware costs
- Lower personnel expenses
- Faster development cycles
- Less maintenance work

Running TiDB means that your hardware and maintenance costs go down and the productivity of your team goes up. As the One Database to Rule Them All, TiDB meets your needs without compromise.

Additional resources

- TiDB Documentation
- TiDB case studies
- TiDB white papers and solutions
- TiDB education

Stay tuned with us

- Website: https://www.pingcap.com/
- LinkedIn: https://www.linkedin.com/company/pingcap/
- Twitter: https://twitter.com/PingCAP
- Youtube: https://www.youtube.com/c/PingCAP