Materialize

WHITEPAPER

OLTP Offload: Optimize Your Transaction-Based Databases

The Operational Core of Your Business

The OLTP (online transaction processing) database is so ubiquitous, we hardly notice it. Every day, millions of consumers swipe credit cards, withdraw bank funds, order inventory, and perform other essential transactions. These actions seem unremarkable to us. But without them, the global economy wouldn't function.

In recent years, much commentary has been devoted to data fabrics, LLMs, and other facets of the modern data stack. A victim of its own success, OLTP has generated comparatively little discussion. However, data teams can gain more by optimizing OLTP than by dabbling in the latest data trends.

One of those key optimizations is OLTP offloading. OLTP offload — also referred to as query offload — is when expensive queries are moved off of an OLTP database and performed on other data systems. This improves performance and stability, cuts costs, and preserves systems of record.

Many teams turn to read replicas and other bandaids to execute OLTP offload. But these stopgaps are not durable in the long-term. To perform effective OLTP offload, teams need an operational data warehouse.

Read on to learn everything you need to know about OLTP offload, including all the different methods leveraged by teams. And find out why an operational data warehouse is the right solution.

OLTP Database: Definition, Architecture, Use Cases

Online **transaction** processing (OLTP) is a type of data processing that enables the rapid execution of database transactions. A database transaction is a change, insertion, deletion, or query of data in a database.

OLTP databases must always be available, correct, and low latency to keep a business operational. "Online" means that these database transactions occur very quickly — often in milliseconds. This is why OLTP is ideal for transactions such as purchases, reservations, and deposits.

Developers build services that write to and read from OLTP databases. These services power the core operational tasks of a company. OLTP systems are typically used by frontline workers, such as cashiers, bank employees, and restaurant servers. OLTP databases are also employed for self-service applications, including online banking, e-commerce, and concert ticketing services. View the table below to learn about other key features of OLTP.

OLTP: Key Features

Rapid processing	Simple transactions	Multi-user accessibility	Continuous availability	Indexes for fast data access
Benchmarked by the number of transactions that can occur per second.	Built for basic queries (i.e. calculating bills), insertions, updates, and deletions.	Accessible by multiple users at the same time, but cannot change the same data at the same time.	Continuously available, without downtime, to serve core business operations.	Retrieves and queries data quickly with database indexes.

Another popular database system is OLAP (online analytical processing). OLAP is built for data analysis, while OLTP is used to power business operations. OLAP databases help generate business insights. However, OLTP databases actually run the business processes that allow a company to function, such as payment systems. See the table below for a side-by-side comparison of OLTP and OLAP.

OLTP vs. OLAP

Rapid processing	Simple transactions	Indexes for fast data access
Design	Standard RDMS	Multi-dimensional schema
Response time	Milliseconds	Seconds, minutes, or hours
Type of data	Operational	Primarily historical
Size of data sets	Small	Massive
Type of queries	Basic insertions, updates, deletions, queries	Complex, read-intensive
Workloads	Reads and writes	Reads, no writes (cannot update current data)
Usage	Transaction systems	BI & analytics

Companies use OLTP to support their critical business processes, and OLAP to generate business intelligence from historical data. OLTP enables businesses to operate, while OLAP allows business leaders to make decisions.

The Problem: Running Complex Queries on OLTP

OLTP databases are best suited for simple queries. These simple queries leverage a small number of database records to perform basic transactions, such as calculating account balances. This reduces the workload on the system. Nevertheless, there are often cases when performing complex, read-intensive queries on an OLTP database might end up occurring.

As an example, consider a mortgage underwriting company. In this scenario, the mortgage company needs to use borrower data to determine if an applicant qualifies for a mortgage. The company's OLTP database captures operational data from front end systems, online loan sites, and backend databases.

However, the underlying queries that calculate loan eligibility are complicated, requiring joins and other complex SQL instructions. The OLTP database slows down and destabilizes when performing complex queries that join across tables, filter across time windows, and compute aggregations. As a result, mortgage determinations take longer, causing the company to fund less loans.

This is how OLTP systems can easily strain due to read query-intensive workloads. As teams start to run more complex queries on OLTP databases, the services offered by the system begin to deteriorate. As a result, teams need to scale up their operations. They must buy more software and hire more workers. This is expensive, but often unavoidable, since OLTP systems are mission-critical.

To relieve pressure on the core database of record, data teams need to find another approach to OLTP. They must protect the core database and the business on which it relies. And one way to do this is by offloading expensive (i.e. high compute) queries from OLTP databases. This is the rationale for OLTP offload.

The Solution: OLTP Offload

Since OLTP databases are essential for business operations, running complex queries on them can slow down critical processes, including payments and transactions. These complex reads in the OLTP system can create strain, leading to downtime, missed latency SLAs, and excessive costs

OLTP offload offers a solution. With offload, teams transfer expensive, complex queries off of OLTP databases. Instead, they perform these queries on read replicas, separate databases, or other data platforms.

This allows OLTP systems to deliver reliable and performant services without strain from read-intensive query workloads. OLTP offloading offers teams a number of advantages, including:

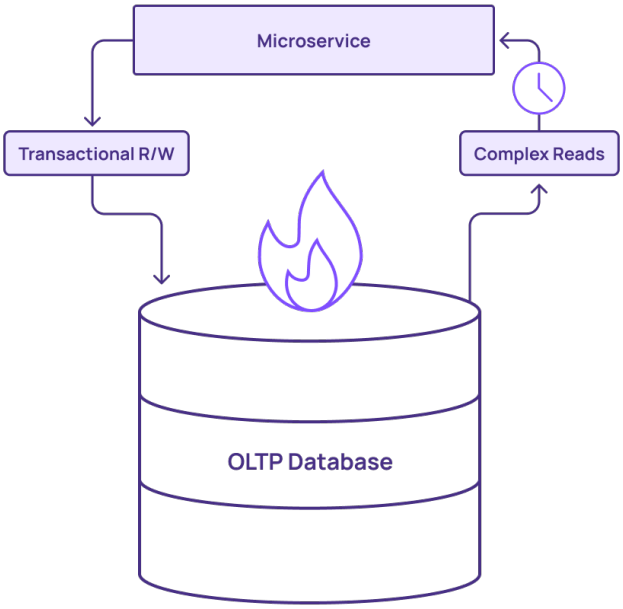
- **Fresh, responsive results** - By removing complex queries from the OLTP system, the database can perform transactions faster. Data freshness is preserved, and the results are correct.
- **Improved reliability** - When teams remove expensive queries, they lessen the strain on OLTP databases. This eliminates destabilization and downtime, which can cause lost revenue and support for a brand or service.
- **Cost savings** - OLTP offload can save teams money when performed on data platforms that are more accommodating of complex queries.

Many OLTP offload methods do not realize all three of these advantages. Teams need an operational data warehouse to satisfy all three. An OLTP offload could arise for many different use cases. But some standard ones include:

- **Queries from dashboards or UIs taxing the database** - Dashboards and front-end interfaces query OLTP databases upon user request at unpredictable intervals. The random execution of complex queries can overwhelm OLTP systems.
- **Teams writing services/jobs that take data from the OLTP system, denormalize it, and write it back** - Denormalization jobs take data from the core OLTP system, processes, and writes it back to reporting tables. This can be done in batches, or incrementally on write via techniques like triggers. In either case, this takes up developer time and adds complexity. This also increases the chances of bugs since developers are doing work that SQL would normally do.
- **The presence of materialized views** - Materialized views store query results in memory. This is effective for denormalizing data, but the results are not fresh. As new data enters the database, materialized views are not automatically updated. They must be re-run in order to reflect current results. This constant query recomputation is expensive.
- **The presence of read replicas** - Teams use read replicas to query data, so they can take the load off core OLTP databases. However, read replicas are still not optimized for complex queries, since they mirror the OLTP architecture. There is still a lag in generating results. As such, read replicas do not result in fresh, responsive results.

In practice, there are several different workarounds for executing complex OLTP queries. Each workaround comes with its own drawbacks. In the following section, we will examine the pros and cons of each workaround.

Workaround #1: Perform Queries On Core Database



Higher Stability	Fresh Results	Low Costs
✗	✗	✓

First, there's the option of performing the queries on the core OLTP database itself. In this scenario, complex queries are run directly on the OLTP database.

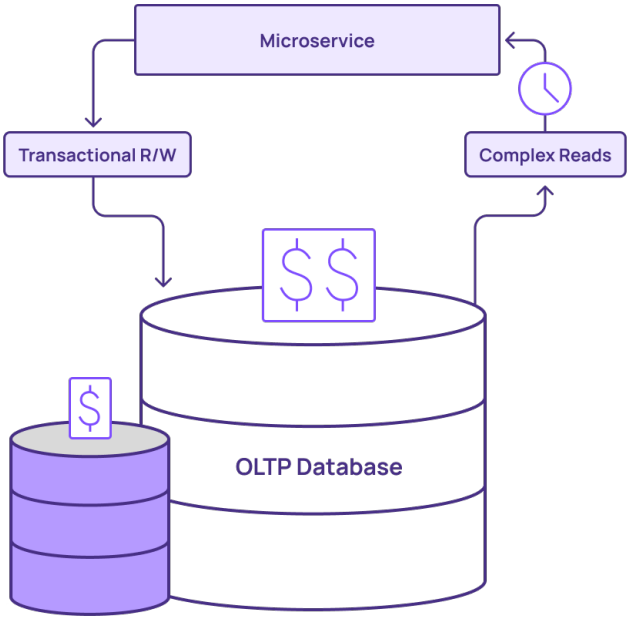
The core database that handles the operational read and write workloads also handles the more expensive analytical queries. No effort is made to offset the impact of high compute workloads on the database.

This can lead to major issues, including:

- Additional indexes to support faster complex reads mean longer write operations, since each index must be updated on write.
- Denormalization jobs, taking data from the core OLTP and writing back to reporting tables, takes up developer time and adds complexity.
- Materialized views can essentially do the up front denormalization work, but that comes at a cost when the view is refreshed. Fresher data means more load on the database.

When you perform queries in-place, data freshness suffers, UIs won't match customer actions, and reports are out of date. Updating the materialized view also creates load on the database. As you recompute views more frequently to get fresher results, you're basically just re-running queries constantly.

Workaround #2: Scale OLTP Database



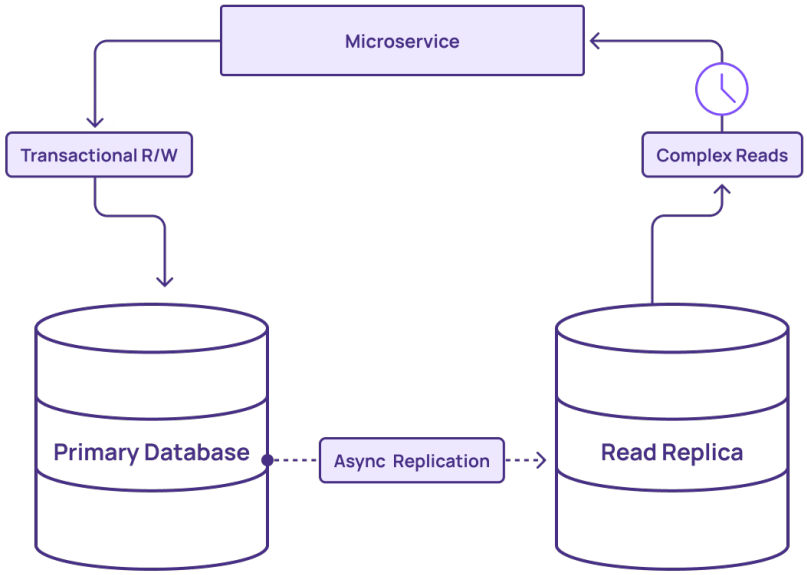
Higher Stability	Fresh Results	Low Costs
✅	❌	❌

Once teams have exhausted the resources of their OLTP database, they might choose to scale up to a bigger machine. Bigger machines are better able to handle the query load. This could lead to more reliable service and less downtime. However, the queries are still being performed on the core database of record, and this can result in a number of issues:

- Databases are not cheap to scale up. The price/performance ratio in regards to the complex query might be unfavorable.
- Complex, high compute queries still take longer to perform on the OLTP architecture, slowing down services.
- Teams eventually reach a hard limit on how much they scale their database.

By simply scaling up databases, teams can reconcile some stability problems, but high latency can still become an issue. As a result, data freshness suffers, and results are stale. Additionally, scaling up machines can become expensive quickly as the demands of complex queries continue to rise.

Workaround #3: Read Replica



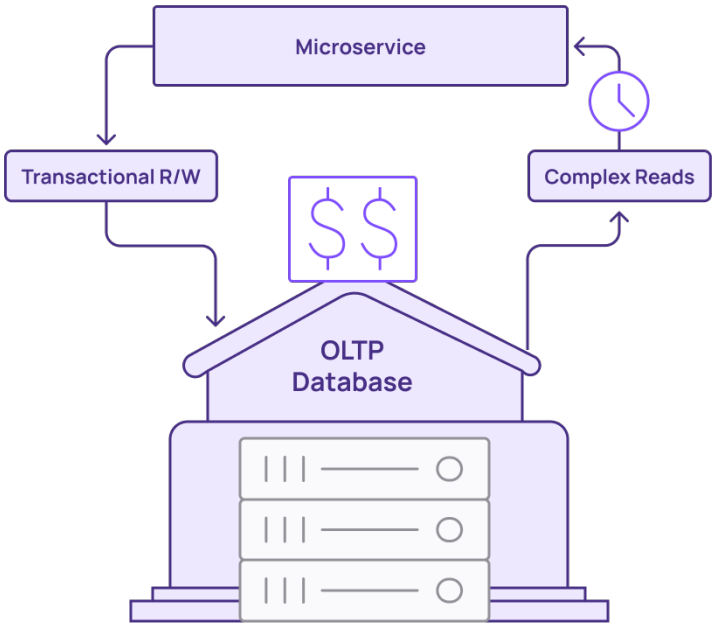
Higher Stability	Fresh Results	Low Costs
✅	❌	❌

Read replicas are the common method for offloading read-heavy workloads from an OLTP system. By replicating the database to one or more read-only copies, businesses can distribute the read load and alleviate stress on the primary OLTP database. However, this approach comes with significant trade-offs:

- The replicated database is still using the same architecture as the primary. So if the primary could not return a complex aggregation in a fast enough time, the replica may not be able to either. You could scale up the read replica, but that introduces cost and still might provide services too slowly.
- You can store many indexes on the read replica. This will help speed up the queries, but the queries will still take longer to return as the data size grows. Writes become slower as a result. This slows down the primary database if the replication is synchronous. You can configure the replication to be asynchronous, but then you must contend with eventual consistency.
- Read replicas increase infrastructure costs, and the ROI may not be there. Each replica consumes storage and computational resources, which can become expensive as the number of replicas grows. If you can't get high utilization of these replicas, you may be wasting resources.

Read replicas can help relieve the load from OLTP systems, increasing database stability. But because read replicas share the same design as the original OLTP database, they still perform queries with high latency. Services can remain slow, even though teams spend more money on hosting the read replica.

Workaround #4: Analytical Data Warehouse



Higher Stability	Fresh Results	Low Costs
✓	✗	✗

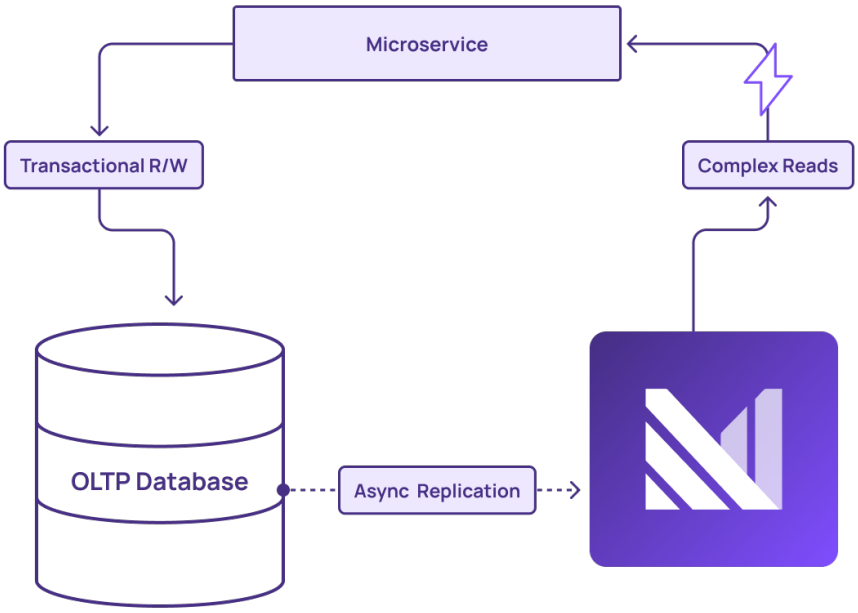
Complex queries are not ideal for OLTP systems.l, but analytical data warehouses are built for these kinds of queries. Teams can perfect SQL logic in their analytical data warehouses using historical data. It's also not uncommon for teams to use analytical data warehouses to perform OLTP offload.

Although analytical data warehouses offer more stability than OLTP systems, they also come with their own limitations:

- Analytical data warehouses eventually reach a hard limit on data freshness. They run on batch processing, and to approach the freshness needed for OLTP, they must constantly run data updates. Although this generates fresher data, the data updates can only occur so fast. This leads to inadequate data freshness.
- Operating in a pay-per-query pricing scheme, analytical data warehouses can generate high costs when performing OLTP offload. Constantly re-executing queries and updating data for fresh results can create a growing cost center.

Analytical data warehouses can handle the complex queries that overwhelm transactional systems. But issues with data freshness and costs make them a less appealing choice for OLTP offload.

Materialize: Operational Data Warehouse for OLTP Offload



Higher Stability	Fresh Results	Low Costs
✓	✓	✓

By offloading the queries from an OLTP system to Materialize, organizations can improve the resilience and performance of their core services while ensuring fast and fresh query results. Materialize enables a more efficient and reliable data handling process, keeping core operations smooth and responsive.

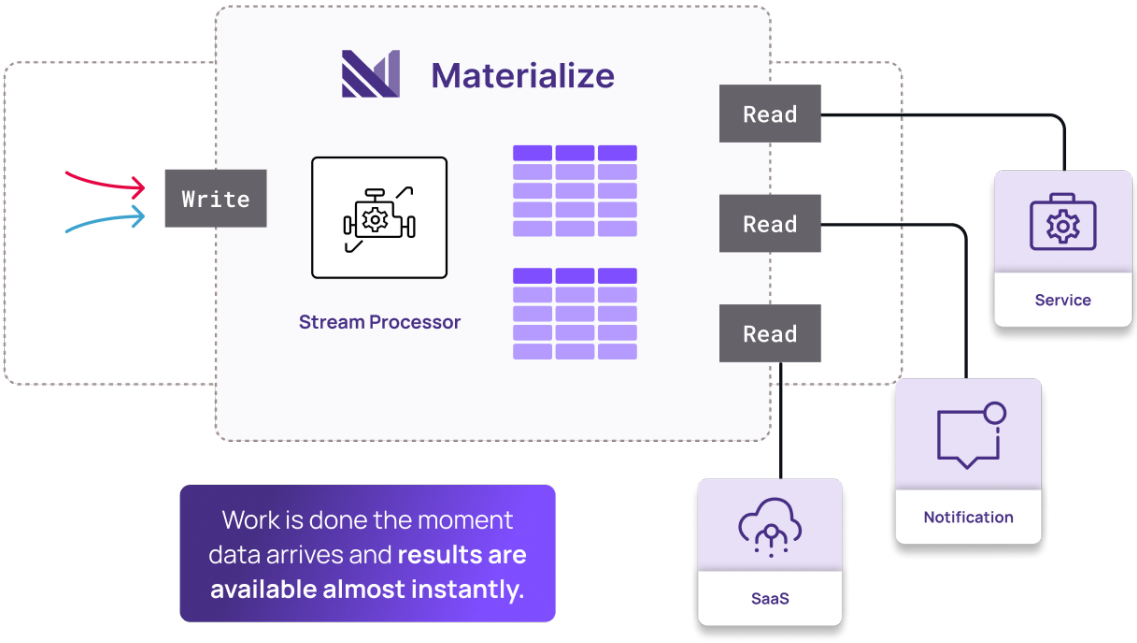
Materialize combines the power of streaming data with SQL support. With Materialize, teams can access the familiar interface of a data warehouse, but powered by fresh data, as opposed to batch data warehouses that update data every few hours.

Materialize achieves the low latency necessary to achieve parity with the speed of OLTP systems. Results are returned by Materialize in milliseconds, the same range as OLTP databases. This ensures business operations and transactional systems remain unbroken. Conversely, batch data warehouses return results in seconds, minutes, or hours, too slow for a transaction-based system of record.

As a data warehouse, Materialize is also able to handle complex queries. One of the benefits of the data warehouse architecture is its ability to perform complicated joins and aggregations across millions of records. However, traditional data warehouses cannot perform these queries over fresh data, meaning the results are out-of-date. But Materialize executes arbitrarily complex queries over streaming data, meaning the results are fresh enough to be used in OLTP workflows.

When OLTP databases experience reliability issues, consistency can suffer, and incorrect results can be recorded in the system. Materialize adheres to strong consistency, meaning that results always match the corresponding data inputs. Materialize also offers real-time recency. This guarantees strong consistency, even with aysnc replication. Materialize's consistency guarantees mean that the results of complex queries are always correct, matching the accuracy needed for OLTP systems.

Materialized views are sometimes used on OLTP systems to denormalize data and commit it to memory, where it can be accessed repeatedly. However, materialized views are not automatically updated, meaning the data is not fresh, and insufficiently up-to-date for OLTP transactions. Leveraging Incremental View Maintenance (IVM), Materialize incrementally updates materialized views as new data streams into the system. This limits the amount of work the data warehouse does, and allows materialized views to always stay up-to-date.



Materialize incremental computation engine
Work done on write, reads are fast and computationally free.

This is how Materialize decouples cost from query freshness. Teams can harness materialized views in Materialize to perform complex OLTP queries at a fraction of the cost. At the same time, the requisite data freshness is maintained for OLTP transactions.

With Materialize, teams can implement a Command Query Responsibility Segregation (CQRS) pattern that sends the writes to the core database, and the reads to Materialize. This allows teams to save money by scaling down their main database. This also leads to happy customers, due to extremely fresh views. No matter how popular these reports become, the core database can always keep up.

When teams do not want to introduce a new service to call, they can expose the view supporting their app as a table directly in the database using a postgres feature called a foreign data wrapper. Now they get all the benefits of Materialize, without requiring their app to directly call a new service.

With Materialize, teams can offload complex queries from their OLTP systems, and cost-effectively perform them with millisecond latency. This allows OLTPs to execute simple read/write operations, without straining the transaction systems. Materialize handles the complicated reads that would otherwise negatively impact the performance and reliability of the OLTP database.

Offload Your Expensive OLTP Workloads onto Materialize

OLTP databases are the technological core of the world economy and other mission-critical systems. They perform the transactions that companies rely on to keep their basic business operations functioning.

OLTP systems are built for simple queries that handle insertions, updates, and deletions. But as a rich store of operational data, OLTP databases inevitably inspire complex queries. However, complex queries are expensive, and negatively impact performance, reliability, and data freshness.

Materialize empowers you offload your expensive OLTP queries onto a real-time data warehouse. This allows you to perform complex queries on fresh data at a fraction of the cost, enabling you to successfully offload expensive workloads from OLTPs, without breaking operational workflows.

> **If you're running complex queries on OLTP systems, try Materialize for free now to offload your expensive workloads.**