Implementing an Operational Data Layer for Product Catalog Modernization

Why retailers are moving from PIMS to an operational data layer to meet product catalog requirements

September 2022

Content

Introduction

The product catalog is at the heart of any retail company- simply put it is the window into the list of all products that are for sale and their information. These days that information can be extensive- from name, dimensions, price (and per-store price) to images, videos, and third party reviews. For a seamless digital, in-person or omnichannel experience, the data needs to be consistent and available in real-time for a number of different retail front ends- the ecommerce website, the customer facing application, in-store workforce applications, POS devices and more.

Product catalog data management is a complex problem for retailers. Older, more established retailers have traditionally relied on vendor-supplied product information management systems (PIMS) as their backend data store. The PIMS is a tool or set of tools built specifically to help retailers produce and manage product content across the various digital channels they support. Typically, this will act as the “single source of truth” across the organization for product information, including technical data, usage data, emotional data and digital catalog data (number of products, variations, seasonality).

Product data will be input to the PIMS and then a subset of it will be ETL’d to a product catalog data store to be combined with other necessary information that the viewer will require. For example, the product catalog for ecommerce will need review data and upsell recommendations, whereas the product catalog for the workforce mobile app will need in-store stock location information. The rigidity of the relational database management systems (RDBMS) used often meant that a new product catalog data store was created on top of the PIMS for each use case. This led to a proliferation of data silos that each needed to be updated every time a new product or product information was added, and unreliable consistency when it comes to offering an omnichannel experience as stock and location of items is near impossible to reconcile.

After years of relying on PIMS and multiple monolithic, vendor-provided systems, retailers have learned that product catalogs built on legacy databases are unsuitable for modern ecommerce experiences. Retailers are now increasingly approaching their product catalog modernization by implementing an operational data layer (ODL) that centrally integrates and organizes siloed product data, making it available to all consuming applications. This enables a range of use cases such as single view of product, real-time analytics, and omnichannel experiences. Here we’ll discuss the modernization drivers, the journey to implement an ODL and why MongoDB’s document model and architecture make it a great fit for the use case.

Modernization Drivers for Product Catalog

Let's discuss in more detail some of the challenges that arise from the PIMS approach that retailers are using today. Broadly speaking, the data silos and the complexity of keeping them consistent and up to date cause a lack of access to rich, consistent product information across sales channels, which is vital for a smooth customer journey.

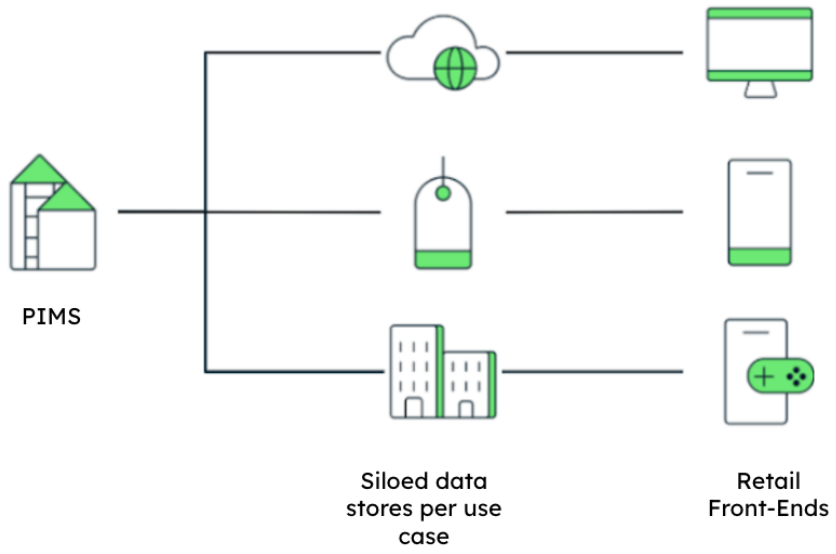

Figure 1. Current PIMS approach shows siloed data stores

Common Challenges with Legacy Product Catalogs

Loss of revenue and customer trust due to inconsistent view of product

- Digital, in-person or omnichannel journeys (e.g. buy online, pick up in store) require an up to date view of product and stock to prevent risk of loss of sale and customer trust.
- There are inconsistencies in product data resulting from siloed data or the spaghetti architecture that most retailers are forced to put into place to bridge the gap between their legacy estate and the modern front end applications. Different data stores meet different uses for the data (e.g. analytical vs. operational) and a

tangled web of ETL processes between them- this creates complex data synchronization issues and lags in updating data.

Loss of competitive edge due to delays in new products and info going live

- Delays in bringing products to market quickly can cause a loss of competitive edge (e.g. fast fashion: SHEIN uploads 6k products a day to keep up with trends), and the inability to add information to existing products can lead to delays in the overall business's initiatives (e.g. grocery: updating food packaging reqs for expansion to new geo).
- Relational data models that require up-front schema definition and downtime to change schema inhibit the ability for new products (with potentially different attributes) to be added or new attributes for existing ones to be discovered. Waiting for change windows and time spent retro-fitting object based data into a tabular schema is a non value add effort that delays launch & update. Additionally, the inability to flexibly scale data infrastructure can lead to slow-downs as data size increases.

Increase in churn or bounce rate due to poor product search capabilities

- Despite the fact that most customers start their digital buying journey at the search bar, it's estimated that 84% of companies aren't optimizing their online search experience. Customers expect an Amazon-esque search experience, and studies show that after a successful product search a customer visit is 2x more likely to convert to a sale.
- Legacy architectures typically require a retailer to maintain at least 3 different systems to enable this capability: 1) data in RDBMS, 2) search technology, 3) caching layer to allow faster responses to page rendering. This is costly to manage and maintain, because developers need to integrate with several different technologies and query languages, which often leads to synchronization issues.

Inability to increase revenue through cross sell or upsell

- Maximizing the spend per customer is one of the best ways to gain additional wallet share (e.g. AO.com % profit from install , insurance and cross sell). This requires the ability to do real-time in-app analytics and provide personalized recommendations.
- Many retailers are stuck in the past- executing overnight batch jobs to output analytical data from their operational database into another system. This causes a lag time in the availability of data to do analytics on and removes the ability to provide real-time personalized promotions or recommendations.

Difficulty maintaining profitability due to low margins and high operational cost

- As competition for customers rises, many retailers are squeezed to lower prices and operate on a thin profit margin, e.g grocery chains that price match their competitors. For sustainable business, the cost to serve them must be brought down to match.
- Maintaining and managing a legacy tech estate can have high operational costs, especially compared to cloud service based offerings. Additionally, one has to consider the sometimes punitive licensing policies of RDBMS vendors and MIPS charges from Mainframe.

So what does a Modern Product Catalog need?

- Single consolidated view of the holistic product catalog
- Flexibility to model and evolve diverse products over time
- Scalability to add products and product data without sacrificing speed
- Search that is easy to enable and provides fast sophisticated results
- Analytical capabilities to offer personalized up sell and cross sell

These are the tenants of a modern operational data layer (ODL): a single consolidated view across all data that has the flexibility, scalability and technical capabilities to serve the operational and analytical needs of the consuming business applications.

The sample reference architecture of this pattern is visualized in the diagram shown below. An operational data layer (ODL) is an intermediary between existing data sources and consumers that need to access that data.

By deploying an ODL in front of legacy systems, new business initiatives and requirements can be met without the risk or difficulty of ripping and replacing the existing architecture. It can reduce workload on source systems, improve availability, reduce end-user response times, combine data from multiple systems into a single repository, serve as a foundation for re-architecting a monolithic application into a suite of microservices, and more. The operational data layer becomes a system of innovation, allowing the business to take an iterative approach to digital transformation.

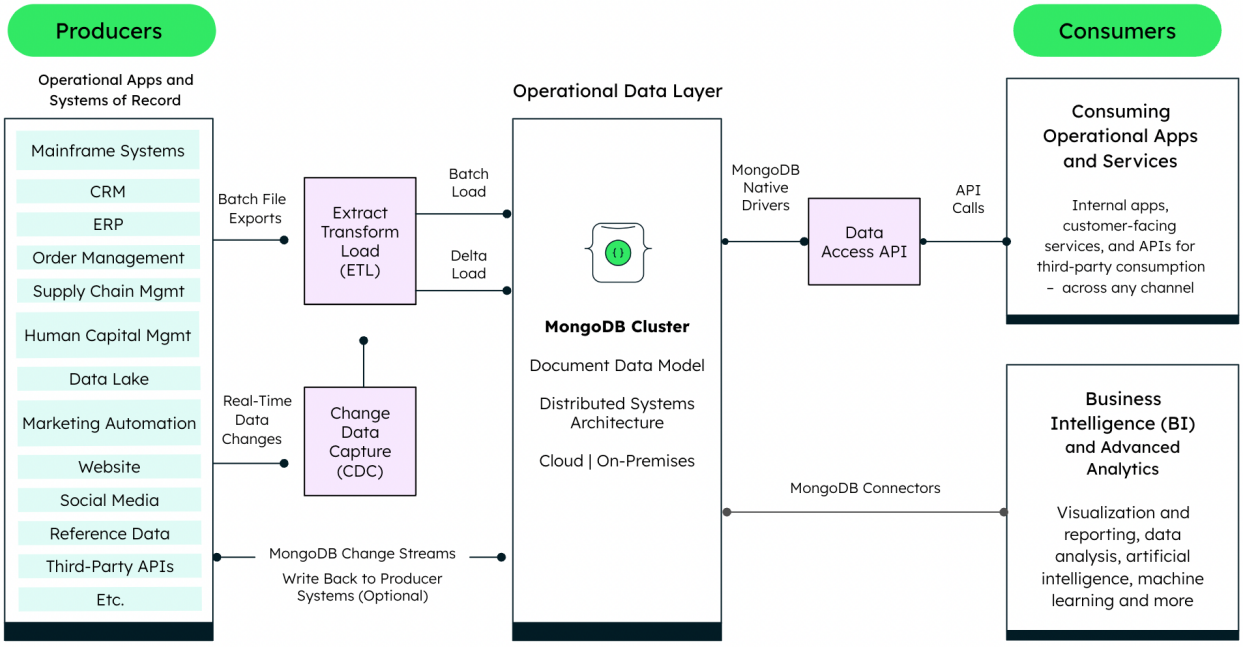

Figure 2. Producer and consumer communication via MongoDB's Operational Data Layer

Architecting an Operational Data Layer for Product Catalog

The implementation of this architecture tends to be a staged approach, with degrees of value attributed to each stage. At each stage of maturity along this journey, we'll see added business benefits to the product catalog.

Building an Operational Data Layer

Outside of the ODL itself, the chief components in this architecture are the source systems and the consuming systems. In the move towards an ODL architecture, the transfer of data from the source systems or “Data Producers” is the first step. These are usually databases, but sometimes file systems or other data stores. Generally, they are systems of record for one or more applications, either off-the-shelf packaged apps: PIMS as mentioned above is one of the key sources, but data will also be coming from, ERP, CRM, etc. as well as internally-developed custom apps. or companies with an earlier established data estate this could be a Mainframe.

Data loading will typically take two stages: Initial day one bulk loading of data and then delta updates of the additional products added or product info changed since the last load.

Bulk Loading

For the initial day one bulk load there are a wide range of methods and technologies that can be used, but this a classic Extract Transform Load (ETL) function- the data must be taken from the source database, transformed into a format suited to the operational data layer, and then loaded into the ODL.

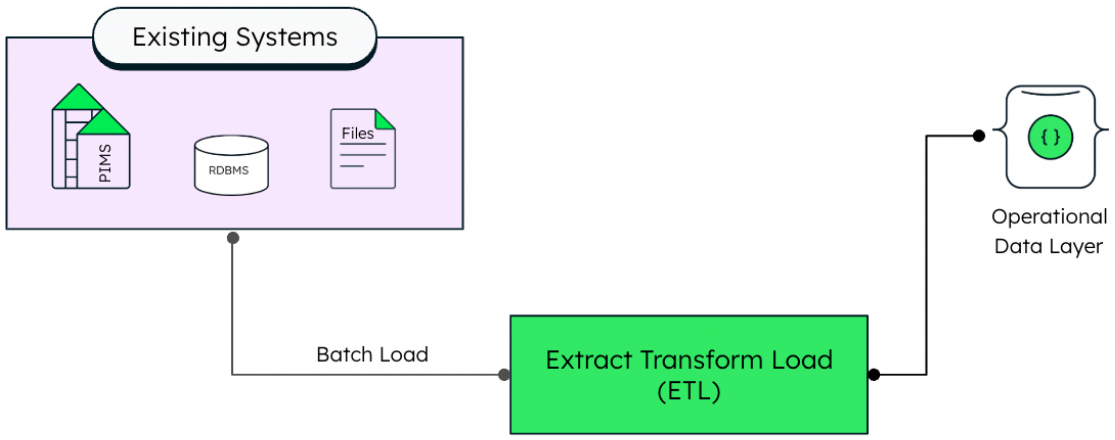

Figure 3. Bulk loading from heritage system to operational data layer via ETL process.

It's key here to remember that the movement of the data is only one part- the transformation of the data to a new model that is suited to provide optimum performance for the intended workload is key. For a product catalog use case, the schema should be optimized for the APIs that will be driven from the ODL and the performance of the most business critical queries from the new microservices/ applications that will be developed. For most product catalogs, the primary workload will be read dominant, so the data modeling should reflect this. This transformation can be done en-route (most common) or data could be landed in a staging database and transformed in the MongoDB ODL itself using aggregation (in this instance it must be ensured that the data is synced correctly).

Some data modeling and architecture best practices for product catalog are covered [here]. MongoDB Data Modeling courses are available at MongoDB university [here]. The use of MongoDB Professional Services to provide expert opinion, training & MongoDB Relational Migrator is recommended.

Data can also be moved and transformed with custom scripts or batch jobs built in-house, or with off-the-shelf ETL tools. Some of the most commonly used off-the-shelf ETL technologies include Informatica, Talend, Ab Initio, IBM Datastage and Pentaho.

Delta Loading

The product catalog now needs to be kept up to date with new products that are added to the existing systems and changes or additions to existing products. For other ODL use cases this "Delta Load" could be done in a batched manner- end of the day or end of the week, but for a modern product catalog this should be kept up to date in real-time, so that any read workload gets an accurate view of the product.

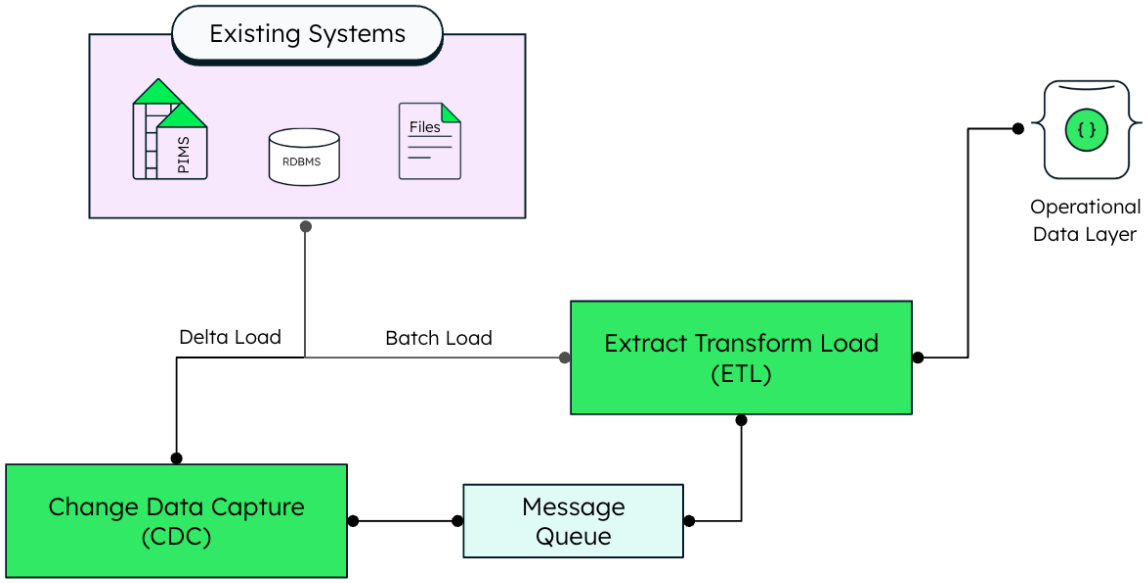

Figure 4. Delta loading process

Change Data Capture in this instance is most commonly achieved through the use of Kafka. Many source databases will have a Kafka connector, which can act as a source (e.g. Oracle GoldenGate has a BigData handler for Kafka) to publish data to topics, which the MongoDB Kafka connector can then sink to the ODL.

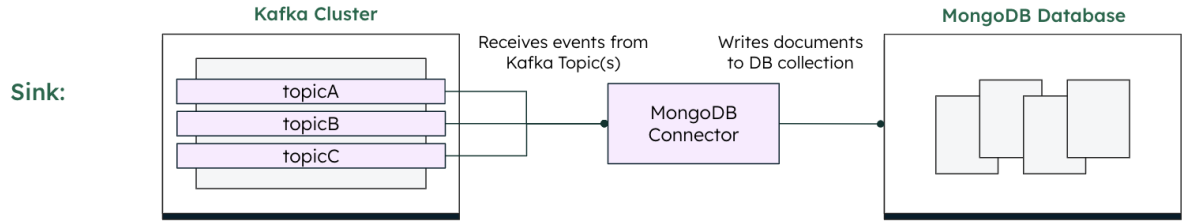

Figure 5. Leveraging MongoDB's Kafka Sink Connector for Change Data Capture

Depending on the tech stack and skill sets that are already present in the organization, other technologies are also adopted for this purpose, for example, other messaging technologies like MQTT or RabbitMQ, Pub/Sub technologies e.g. Google Pub/Sub & Dataflow or an off-the-shelf CDC solution like Qlik Replicate (Attunity) or Debezium.

At this stage, the bulk of the read workload can now be redirected to the ODL rather than the legacy data stores. This will have an immediate impact on the speed and stability of the product catalog. In the next section we'll talk about additional benefits that can be found by maturing the Product Catalog and what the next stages of leveraging the architecture shift are. It's important to note that even if the architecture shift ends here, there is huge value to be found in this alone, including:

- **Speed**: By serving product catalog read queries from a highly performant system whose architecture is optimized for read workload, UI speeds will be improved leading to better customer and employee experience with a knock on effect into revenue generation and efficiency of workforce.
- **Stability**: MongoDB's replica set architecture typically provides enhanced uptime (esp Atlas 99.995% SLA).
- **MIPS Reduction**: In the case where the source system is mainframe, MIPS will be lessened as the reads will hit the ODL instead.

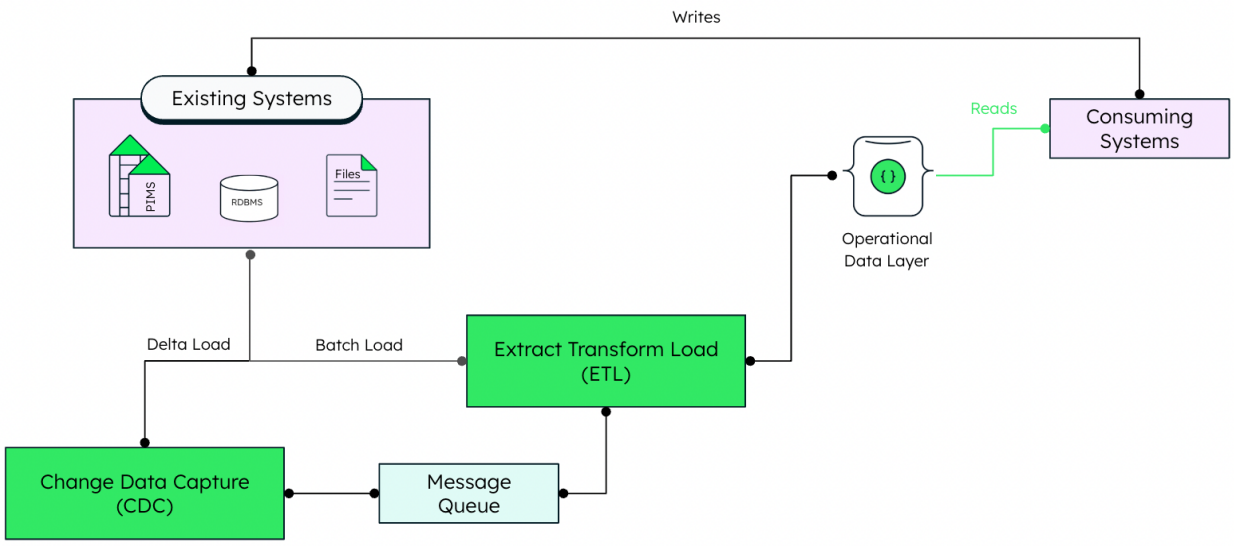

Figure 6. Read queries are directed to the ODL while writes go to the previously existing systems

Maturing the Product Catalog

The creation of the operational data layer for the product catalog can be the start in a journey to move away from legacy tech all together. Retailers will begin to build functionality, analytical insights and use the product catalog as a point to begin to redevelop existing applications. The maturity curve in retailers follows a staged approach, transforming the role of the data layers.

Often, an operational data layer evolves over time. From a focused and simple start, it grows in scope and strategic importance, delivering increased benefits to the business. In fact, it is a best practice to begin an ODL implementation with a limited scope and grow it over time. An operational data layer that tries to incorporate all possible data sources from the beginning may stall before it proves its value; it's much better to demonstrate the ODL's capabilities with a small set of sources and consumers, then iterate from there, incorporating best practices that have been developed along the way.
One of our observations of working with many enterprises on their ODL projects is that as more data sources are integrated, the ODL becomes valuable for serving reads from a broadening range of consumers. Over time, it begins accepting writes, and ultimately can become a system of record in its own right.

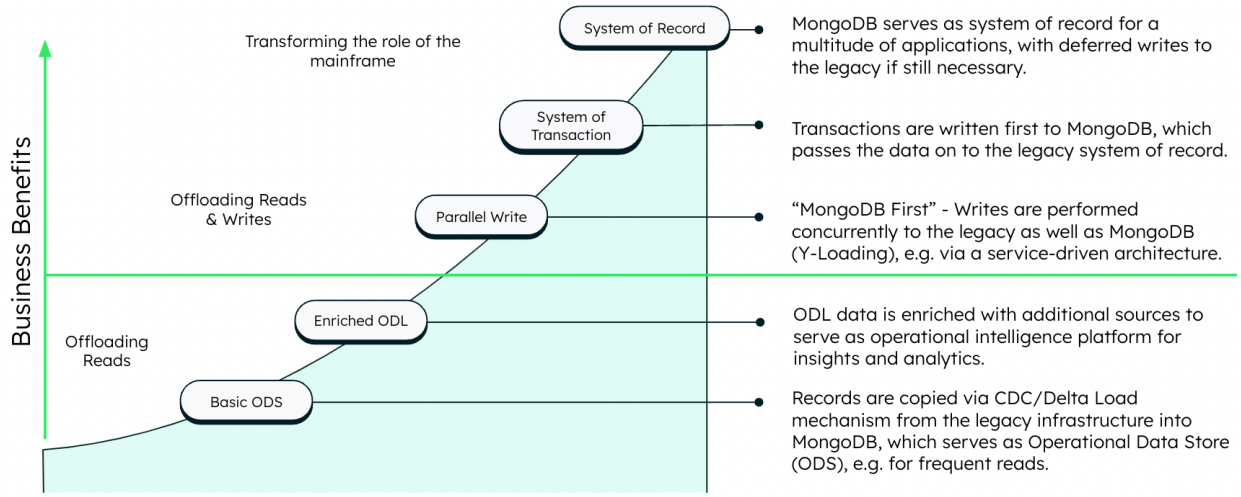

Figure 7. Maturity Model of an Operational Data Layer

Phase One: Creating the Product Catalog ODL

This phase is covered in the above section- by moving and combining records from legacy existing systems to an operational data layer, the read workload of the product catalog can be served in a more performant and resilient manner.

Phase Two: Enriched Product Catalog

Once the ODL has proven its value, a logical next step is to enrich its data by adding useful metadata or integrating new (related) data sources. With this enrichment, the ODL can not only offload more reads from source systems, but also enable use cases that weren’t possible before. A typical use case for this is to enable advanced analytics across a fuller picture of data or create a single product view.

German retailer OTTO combined customer preference data into their product ODL to provide a personalized ecommerce experience https://www.mongodb.com/customers/otto.

Phase Three: Offloading Reads & Writes

The ODL's scope can be expanded by introducing a smarter architecture to orchestrate writes between both source systems and the ODL concurrently. In this phase, when a given consuming system performs a write, it goes to both an ODL and a source system, either directly from the application or via a messaging system, API layer, or other intermediary from which both repositories can receive the write. This pattern is also referred to as "Y-loading", and can help lay the foundations for a more transformational shift of the ODL's role in your enterprise architecture.

Some organizations move directly to phase 4 below, but Y-loading can allow you to run both systems in parallel and road-test the ODL before using it as the primary system for writes.

This allows retailers to build new application functionality making use of the enriched product data and features that the ODL can provide.

- Ticketek Combines MongoDB Atlas and Google Cloud to Drive Analytics
- Queenly Create a Robust Search Engine for Formalwear with MongoDB Atlas Full Text Search

Phase Four: ODL First/ System of Transaction

In this phase, all writes are directed to the ODL by default. Where necessary, changes are routed back to the older data stores, either so that other legacy applications can continue to rely on a source system before being redirected to the ODL, or merely as a precautionary fallback. The secondary write to the source system can be accomplished with a CDC system listening to the ODL or a similar system, like MongoDB Triggers.

- AO: Discovering Customer Signals In Real Time

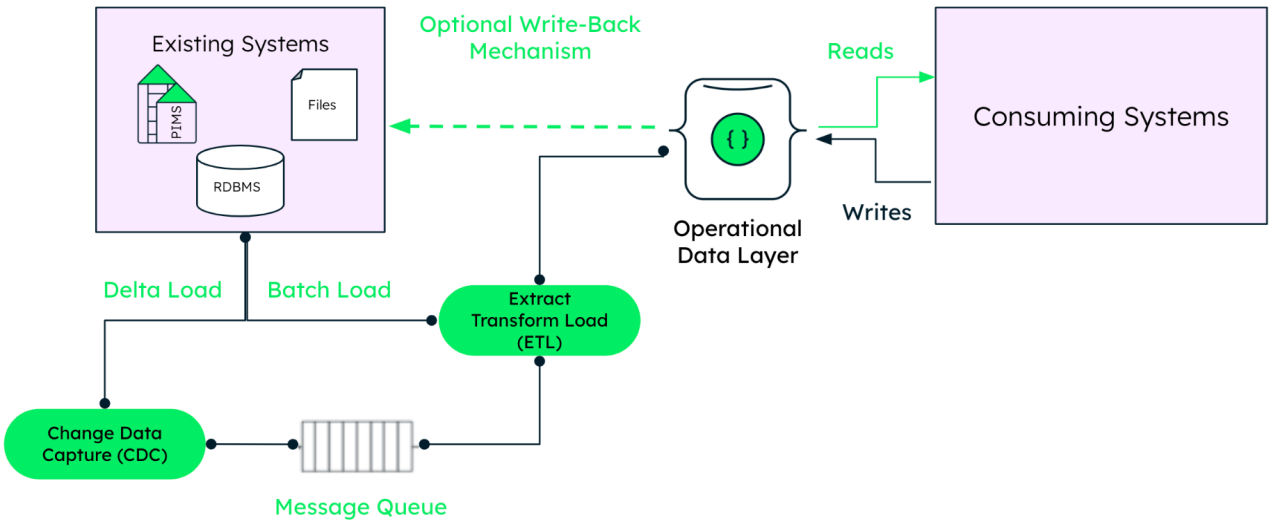

Figure 8. Operational Data Layer, accepting writes and optionally pushing them back to source systems, with select source systems decommissioned

Phase Five: System of Record

Ultimately, the operational data layer can evolve to serve as the system of record. Once all consuming systems for a given legacy source have been ported to the ODL, and its stability has been proven, the source system can be decommissioned for cost savings and architectural simplicity.

Retailers and software vendor solutions are building and replatforming product catalog solutions directly on MongoDB. Commercetools, a leading provider of ecommerce solutions uses MongoDB for its entire catalog management solution. https://www.mongodb.com/collateral/ecommerce-at-mach-speed-with-mongodb-and-commercetools

Why MongoDB for a Product Catalog ODL?

We covered earlier the requirements for a product catalog and how they map nicely to the capabilities of an operational data layer. Now why might a retailer consider MongoDB in this context? Operational use cases are exactly what MongoDB was designed for; the

ability to handle high throughput workloads at speed to serve modern application needs. Naturally this has led to operational data layers becoming the most common architectural application of MongoDB's technology. This is true across all industries, but especially in Retail, where we see standardized adoption of MongoDB to meet use cases like product catalog, single view or customer 360. Let's look at the requirements we discussed above.

Single consolidated view of the holistic product catalog

- **Versatile Document Model:** Documents are a natural way to describe data- in product catalog use cases where descriptions of physical objects are stored, the mapping to a document becomes a natural fit. They present a single data structure, with related data embedded as sub-documents and arrays. This makes it easy to store product data where relationships between products are complex in a way that is increasingly difficult in a row-column structure; think of hierarchies, per-store pricing, recommended product pairings, and parts with dozens of permutations of dimensions. These problems become easy to solve when there's optionality in the modeling process.
- **Rich Query Language:** A single view storage approach like an operational data layer is only successful if it provides the ability for all of the consuming applications to work with the data easily. Many document model databases do not provide hard types (inc. geoJSON or Decimal128), expressive query language operators, or the ability to join data between collections, creating a burden on the application layer. MongoDB's rich and expressive query language provides all of the above and more, including multi-document ACID transactions and the aggregation framework to create sophisticated processing pipelines for data analytics and transformations. For a product catalog that must serve multiple consuming applications- both customer and business facing - it's imperative that data be easy to work with. Similarly, new consuming systems that connect to the ODL will have access patterns and query requirements that haven't been seen before. An operational data layer needs to be versatile enough to meet a wide variety of requirements.

Flexibility to model and evolve diverse products over time

- **Flexible Schema:** With MongoDB, there's no need to pre-define a schema. Documents are polymorphic: fields can vary from document to document within a single collection. For example, all documents that describe products might contain the product ID and the last date it was purchased, but only some of these documents might contain the products it was purchased with or location data from a mobile app. This makes it possible to merge data from source systems storing records on overlapping but non-identical sets of entities. On the consuming side, MongoDB's flexibility makes it easy to alter the data model as needed to meet the requirements of new applications being built on the ODL, for example, no schema change or downtime when adding a product with a new attribute. This flexibility is crucial for an Operational Data Layer. As a product catalog evolves over time, it typically incorporates new source systems, with data model implications that weren't planned from the outset. This can prevent delay in onboarding new products or product lines, or adding additional data to scale the business into new territories which may require new data for local regulations.

Scale to add products & product data without sacrificing speed

- **Speed**: Using MongoDB for an ODL means you can get better performance when accessing data, and write less code to do so. In most legacy systems, accessing data for an entity, such as a customer, typically requires JOINing multiple tables together. JOINs entail a performance penalty, even when optimized – which takes time, effort, and advanced SQL skills. The situation is even worse when you consider that for a given requirement, a consuming system may need to access multiple legacy databases.
- **Unified Data**: In MongoDB, a document is a single place for the database to read and write data for an entity. This locality of data ensures the complete document can be accessed in a single database operation that avoids the need internally to pull data from many different tables and rows. For most queries, there's no need to JOIN multiple records. If the MongoDB-based ODL integrates data from multiple

source systems, the performance benefits of accessing that unified data are even greater.

- **Horizontal Scalability**: Even if an ODL starts at a small scale, you need to be prepared for growth as new source systems are integrated, adding data volume, and new consuming systems are developed, increasing workload. To meet the needs of a product catalog with large data sets and high throughput requirements, MongoDB provides horizontal scale-out on low-cost, commodity hardware or cloud infrastructure using sharding. Sharding automatically partitions and distributes data across multiple physical instances, or shards, all in a completely application-transparent way. To respond to fluctuating workload demand, nodes can be added or removed from the ODL in real-time, and MongoDB will automatically rebalance the data accordingly, without manual intervention.
- **Atlas Online Archive:** As the product catalog grows, there will be a rise in needing to archive off old or seasonal product data, as appropriate. MongoDB Atlas, our fully managed developer data platform, provides the ability to create an online archive, which will automatically "time out" data into cloud object storage to save on costs and increase efficiency. Atlas takes care of the data movement and sync, and provides a unified access API to pull data from both MongoDB and the archive in one query. Imagine a business user wanting to understand a view of all products that are and have been for sale over the past 5 years in a certain category.

Search that is easy to enable and provides fast sophisticated results

The core MongoDB database runs the same everywhere: on-premises in your data centers, on developers' laptops, in the cloud, or as an on-demand fully managed Database as a Service: MongoDB Atlas. Atlas is a true developer data platform that provides additional built-in services to quickly help add features to your application. In the context of product catalog, one stands out:

- **Atlas Search:** MongoDB's Atlas Search allows fine-grained text indexing and querying of data on your Atlas cluster. We have embedded Apache Lucene 8 into the platform to enable advanced search functionality for your applications without

any additional management or separate search system alongside your database. Atlas Search provides options for several kinds of text analyzers, a rich query language that uses Atlas Search aggregation pipeline stages like $search and $searchMeta in conjunction with other MongoDB aggregation pipeline stages, and score-based results ranking. Search is considered table stakes in modern product catalog applications, by implementing a product catalog ODL on MongoDB Atlas, retailers can add functionality like full text search, facets, fuzzy-matching, highlighting, synonyms etc. without considerably increasing their time to market.

Analytical capabilities to be able to offer personalized up-sell and cross-sell

- **In-App Analytics**: Consuming systems aren't limited to operational applications. Within modern product catalogs, retailers are adopting in-app real-time analytics to provide better customer experience and leverage personalized data to provide up-sell and cross-sell functionality. For this, it's necessary to be able to conduct in-place analytics in the operational data layer. MongoDB has unique capabilities to enable this through the ability to isolate workloads- serve analytical queries on up-to-date data on an analytic node of the replica set, without having an impact on production applications. MongoDB's aggregation framework can be used to create advanced processing pipelines to transform and compute data within the database itself.
- **Business Visualization:** The ODL can also provide internal retail teams additional real-time insight into the product data stored; many data analytics tools can use MongoDB's connectors to access the ODL. The connector for business intelligence allows analysts to connect to a MongoDB ODL with their BI and visualization tools of choice; alternatively, MongoDB Charts can connect directly to the ODL for native visualization. The connector for Apache Spark exposes MongoDB data for use by all of Spark's libraries, enabling advanced analytics such as machine learning processes.

Summary

We see retailers embarking on this journey of modernizing their product catalog by developing and implementing an operational data layer. It's a staged approach, with huge amounts of value seen at the first stage, just by the combination of the data. The product catalog ODL delivers a hard to grasp real-time view of products across multiple channels and most importantly faster time to market for new products and services. Implementing an operational data layer is the first, and necessary step towards an enriched product catalog, real time product personalization and analytics, and perhaps a complete move off legacy and into the future.

About the Author

Genevieve Broadhead is the Principal for Retail EMEA in the Industry Solutions team at MongoDB, bridging the gap between new technology trends and how to apply them in a Retail context. Having spent years designing data architectures across industries, she now liaises between product development and some of the largest retailers in EMEA. She sits on the MACH Alliance TECH Council.

Resources

For more information, please visit mongodb.com or contact us at sales@mongodb.com.
Case Studies (mongodb.com/customers)
Presentations (mongodb.com/presentations)
Free Online Training (university.mongodb.com)
Webinars and Events (mongodb.com/events)
Documentation (docs.mongodb.com)
MongoDB Atlas database as a service for MongoDB (mongodb.com/cloud)
MongoDB Enterprise Download (mongodb.com/download)
MongoDB Realm (mongodb.com/realm)

Legal Notice

This document includes certain "forward-looking statements" within the meaning of Section 27A of the Securities Act of 1933, as amended, or the Securities Act, and Section 21E of the Securities Exchange Act of 1934, as amended, including statements concerning our financial guidance for the first fiscal quarter and full year fiscal 2021; the anticipated impact of the coronavirus disease (COVID-19) outbreak on our future results of operations, our future growth and the potential of MongoDB Atlas; and our ability to transform the global database industry and to capitalize on our market opportunity. These forward-looking statements include, but are not limited to, plans, objectives, expectations and intentions and other statements contained in this press release that are not historical facts and statements identified by words such as "anticipate," "believe," "continue," "could," "estimate," "expect," "intend," "may," "plan," "project," "will," "would" or the negative or plural of these words or similar expressions or variations. These forward-looking statements reflect our current views about our plans, intentions, expectations, strategies and prospects, which are based on the information currently available to us and on assumptions we have made. Although we believe that our plans, intentions, expectations, strategies and prospects as reflected in or suggested by those forward-looking statements are reasonable, we can give no assurance that the plans, intentions, expectations or strategies will be attained or achieved. Furthermore, actual results may differ materially from those described in the forward-looking statements and are subject to a variety of assumptions, uncertainties, risks and factors that are beyond our control including, without limitation: our limited operating history; our history of losses; failure of our database platform to satisfy customer demands; the effects of increased competition; our investments in new products and our ability to introduce new features, services or enhancements; our ability to effectively expand our sales and marketing organization; our ability to continue to build and maintain credibility with the developer community; our ability to add new customers or increase sales to our existing customers; our ability to maintain, protect, enforce and enhance our intellectual property; the growth and expansion of the market for database products and our ability to penetrate that market; our ability to integrate acquired businesses and technologies successfully or achieve the expected benefits of such acquisitions; our ability to maintain the security of our software and adequately address privacy concerns; our ability to manage our growth effectively and successfully recruit and retain additional highly-qualified personnel; the price volatility of our common stock; the financial impacts of the coronavirus disease (COVID-19) outbreak on our customers, our potential customers, the global financial markets and our business and future results of operations; the impact that the precautions we have taken in our business relative to the coronavirus disease (COVID-19) outbreak may have on our business and those risks detailed from time-to-time under the caption "Risk Factors" and elsewhere in our Securities and Exchange Commission ("SEC") filings and reports, including our Quarterly Report on Form 10-Q filed on December 10, 2019, as well as future filings and reports by us. Except as required by law, we undertake no duty or obligation to update any forward-looking statements contained in this release as a result of new information, future events, changes in expectations or otherwise.