10-Step Methodology to Creating a Single View of your Business

February 2017

Table of Contents

Introduction

"Single View". "360-Degree View". "Data Hub". A subset of "Master Data Management". Call it what you will – and for the purposes of this paper, we will be calling it "single view" – organizations have long seen the value in aggregating data from multiple systems and channels into a single, holistic, real-time representation of a business entity or domain. That entity is often a customer. But the benefits of a single view in enhancing business visibility and operational intelligence go far beyond understanding customers. A single view can apply equally to other business contexts, such as products, supply chains, industrial machinery, cities, financial asset classes, and many more.

However, for many organizations, successfully delivering a single view has been elusive. Technology has certainly been a limitation – for example, the rigid, tabular data model imposed by traditional relational databases inhibits the schema flexibility necessary to accommodate the diverse data sets contained in source systems. But limitations extend beyond just the technology to include the business processes needed to deliver and maintain a single view.

MongoDB has been used in many single view projects across enterprises of all sizes and industries. This white paper shares the best practices we have observed and institutionalized over the years. It provides a step-by-step guide to the methodology, governance, and tools essential to successfully delivering a single view project with MongoDB.

Why Single View?

Today's modern enterprise is data-driven. How quickly an organization can access and act upon information is a key competitive advantage. So how does a single view of data help? Most organizations have a complicated process for managing their data. It usually involves multiple data sources of variable structure, ingestion and transformation, loading into an operational database, and supporting the business applications that need the data. Often there are also analytics, BI, and reporting that require access to the data, potentially from a separate data warehouse or data lake. Additionally, all of these layers need to comply with

security protocols, information governance standards, and other operational requirements.

Inevitably, information ends up stranded in silos. Often systems are built to handle the requirements of the moment, rather than carefully designed to integrate into the existing application estate, or a particular service requires additional attributes to support new functionality. Additionally, new data sources are accumulated due to business mergers and acquisitions. All of a sudden information on a business entity, such as a customer, is in a dozen different and disconnected places.

Figure 1: Sample of single view use cases

Single view is relevant to any industry and domain as it addresses the generic problem of managing disconnected and duplicate data. Specifically, a single view solution does the following:

- Gathers and organizes data from multiple, disconnected sources;
- Aggregates information into a standardized format and joint information model;
- Provides holistic views for connected applications or services, across any digital channel;
- Serves as a foundation for analytics – for example, customer cross-sell, upsell, and churn risk.

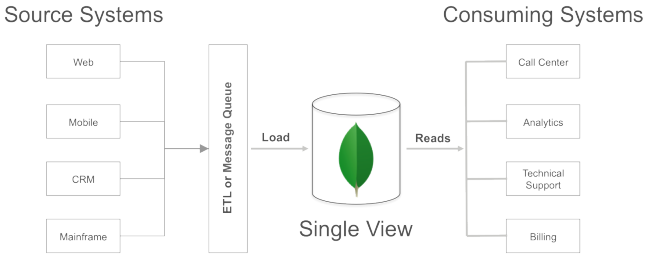

Figure 2: High-level architecture of single view platform

10 Step Methodology to Delivering a Single View

From scoping to development to operationalization, a successful single view project is founded on a structured approach to solution delivery. In this section of the whitepaper, we identify a repeatable, 10-step methodology and tool chain that can move an enterprise from its current state of siloed data into a real-time single view that improves business visibility. Figure 3 shows the methodology.

The timescale for each step shown in Figure 3 is highly project-dependent, governed by such factors as:

- The number of data sources to merge;
- The number of consuming systems to modify;
- The complexity of access patterns querying the single view.

MongoDB's consulting engineers can assist in estimating project timescales based on the factors above.

Step 1: Define Project Scope & Sponsorship

Building a single view can involve a multitude of different systems, stakeholders, and, business goals. For example, creating a single customer view potentially entails extracting data from numerous front and back office applications, operational processes, and partner systems. From here, it is aggregated to serve everyone from sales and marketing, to call centers and technical support, to finance, product development, and more. While it's perfectly reasonable to define a future-state vision for all customer

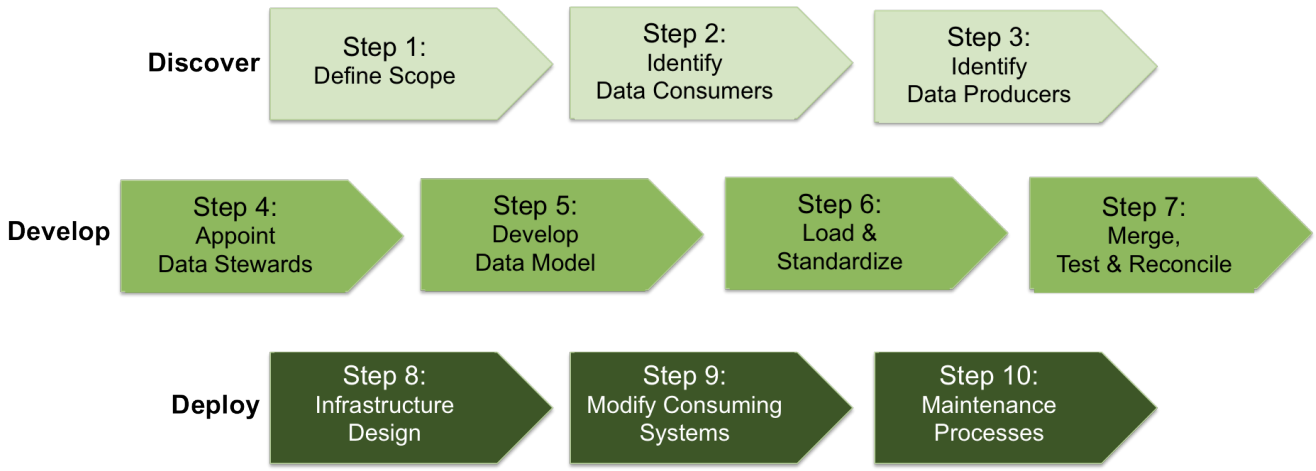

Figure 3: 10-step methodology to deliver a single view

data to be presented in a single view, it is rarely practical in the first phase of the project.

Instead, the project scope should initially focus on addressing a specific business requirement, measured against clearly defined success metrics. For example, phase 1 of the customer single view might be concentrated on reducing call center time-to-resolution by consolidating the last three months of customer interactions across the organization's web, mobile, and social channels. By limiting the initial scope of the single view project, precise system boundaries and business goals can be defined, and department stakeholders identified.

With the scope defined, project sponsors can be appointed. It is important that both the business and technical sides of the organization are represented, and that the appointees have the authority to allocate both resources and credibility to the project. Returning to our customer single view example above, the head of Customer Services should represent the business, partnered with the head of Customer Support Systems.

Step 2: Identify Data Consumers

This is the first in a series of iterative steps that will ultimately define the single view data model. In this stage, the future consumers of the single view need to share:

- How their current business processes operate, including the types of queries they execute as part of their day-to-day responsibilities, and the required Service Level Agreements (SLAs);
- The specific data (i.e., the attributes) they need to access;
- The sources from which the required data is currently extracted.

Step 3: Identify Data Producers

Using the outputs from Step 2, the project team needs to identify the applications that generate the source data, along with the business and technical owners of the applications, and their associated databases. It is important to understand whether the source application is serving operational or analytical applications. This information will be used later in the project design to guide selection of the appropriate data extract and load strategies.

Step 4: Appoint Data Stewards

A data steward is appointed for each data source identified in the previous step. The steward needs to command a deep understanding of the source database, with specific knowledge of:

- The schema that stores the source data, and an understanding of which tables store the required attributes, and in what format;
- The clients and applications that generate the source data;

- The clients and applications that consume the source data.

The data steward should also be able to define how the required data can be extracted from the source database to meet the single view requirements (e.g., frequency of data transfer), without impacting either the current producing or consuming applications.

Step 5: Develop the Single View Data Model

With an understanding of both what data is needed, and how it will be queried by the consuming applications, the development team can begin the process of designing the single view schema.

Identify Common Attributes

An important consideration at this stage is to define the common attributes that must appear in every record. Using our customer single view as an example, every customer document should contain a unique customer identifier such as a customer number or email address. This is the field that the consuming applications will use by default to query the single view, and would be indexed as the record's primary key. Analyzing common query access patterns will also identify the secondary indexes that need to be created for each record. For example, we may regularly query customers against location and products or services they have purchased. Creating secondary indexes on these attributes is necessary to ensure such queries are efficiently serviced.

There may also be many fields that vary from record to record. For example, some customers may have multiple telephone numbers for home, office, and cell phones, while others have only a cell number. Some customers may have social media accounts against which we can track interests and measure sentiments, while other customers have no social presence. MongoDB's flexible document model with dynamic schema is a huge advantage as we develop our single view. Each record can vary in structure, and so we can avoid the need to define every possible field in the initial schema design, while using document validation to enforce specific rules on mandatory fields.

Define Canonical Field Formats

The developers also need to define the canonical format of field names and data attributes. For example, a customer phone number may be stored as a string data type in one source system, and an integer in another, so the development team needs to define what standardized format will be used for the single view schema. We can use approaches such as MongoDB's native document validation to create and enforce rules governing the presence of mandatory fields and data types.

Define MongoDB Schema

With a data model that allows embedding of rich data structures, such as arrays and sub-documents, within a single localized document, all required data for a business entity can be accessed in a single call to MongoDB. This design results in dramatically improved query latency and throughput when compared to having to JOIN records from multiple relational database tables.

Data modeling is an extensive topic with design decisions ultimately affecting query performance, access patterns, ACID guarantees, data growth, and lifecycle management. The MongoDB data model design documentation provides a good introduction to the factors that need to be considered. In addition, the MongoDB Development Rapid Start service offers custom consulting and training to assist customers in schema design for their specific projects.

Step 6: Data Loading & Standardization

With our data model defined, we are ready to start loading source data into our single view system. Note that the load step is only concerned with capturing the required data, and transforming it into a standardized record format. In Step 7 that follows, we will create the single view data set by merging multiple source records from the load step.

There will be two distinct phases of the data load:

1. Initial load. Typically a one-time operation that extracts all required attributes from the source databases, loading them into the single view system for subsequent merging;

2. Delta load. An ongoing operation that propagates updates committed to the source databases into the single view.

To maintain synchronization between the source and single view systems, it is important that the delta load starts immediately following the initial load.

For all phases of the data load, developers should ensure they capture data in full fidelity, so as not to lose data types. If files are being emitted, then write them out in a JSON format, as this will simplify data interchange between different databases. If possible, use MongoDB Extended JSON as this allows temporal and binary data formats to be preserved.

Initial Load

Several approaches can be used to execute the initial load. An off-the-shelf ETL (Extract, Transform & Load) tool can be used to migrate the required data from the source systems, mapping the attributes and transforming data types into the single view target schema. Alternatively, custom data loaders can be developed, typically used when complex merging between multiple records is required. MongoDB consulting engineers can advise on which approach and tools are most suitable in your context. If after the initial load the development team discovers that additional refinements are needed to the transformation logic, then the single view data should be erased, and the initial load should be repeated.

Delta Load

The appropriate tool for delta loads will be governed by the frequency required for propagating updates from source systems into the single view. In some cases, batch loads taken at regular intervals, for example every 24 hours, may suffice. In this scenario, the ETL or custom loaders used for the initial load would generally be suitable. If data volumes are low, then it may be practical to reload the entire data set from the source system. A more common approach is to reload data only from those customers where a timestamp recorded in the source system indicates a change. More sophisticated approaches track individual attributes and reload only those changed values, even keeping track of the last-modification time in the single-view schema.

If the single view needs to be maintained in near real time with the source databases, then a message queue would be more appropriate. An increasingly common design pattern we have observed is using Apache Kafka to stream updates into the single view schema as they are committed to the source system. Download our Data Streaming with Kafka and MongoDB white paper to learn more about this approach.

Note that in this initial phase of the single view project, we are concerned with moving data from source systems to the single view. Updates to source data will continue to be committed directly to the source systems, and propagated from there to the single view. We have seen customers in more mature phases of single view projects write to the single view, and then propagate updates back to the source systems, which serve as systems of record. This process is beyond the scope of this initial phase.

Standardization

In a perfect world, an entity's data would be consistently represented across multiple systems. In the real world, however, this is rarely the case. Instead, the same attributes are often captured differently in each system, described by different field names and stored as different data types. To better understand the challenges, take the example below. We are attempting to build a single view of our frequent travelers, with data currently strewn across our hotel, flight, and car reservation systems. Each system uses different field names and data types to represent the same customer information.

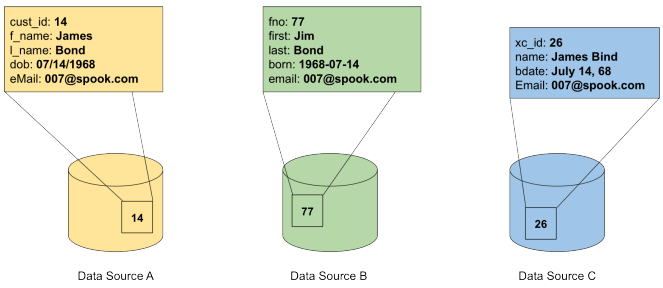

Figure 4: The name's Bond….oh hang on, it might be Bind

During the load phase, we need to transform the data into the standardized formats defined during the design of the single view data model. This standardized format makes it much simpler to query, compare, and sort our data.

Step 7: Match, Merge, and Reconcile

Even after standardizing divergent field names and data types during the data load, inconsistencies can often exist in the data itself. Accurately merging disparate records is one the toughest challenges in building a single view. The good news is that MongoDB has developed tools that can assist in this process.

Looking again at our frequent traveler example above, we can see that the customer names are slightly different. These variances in the first and last names would result in storing three separate customer records, rather than aggregating the data into our desired single view.

It is not practical, or necessary, to compare each customer record to every other customer record loaded from the source systems. Instead, we can use a grouping function to cluster records with similar matching attributes. This should be executed as an iterative process:

1. Start by matching records against unique, authoritative attributes, for example by email address or credit card number;
2. Group remaining records by matching combinations of attributes – for example a last*name, date*of*birth, and zip*code triple;
3. Finally, we can apply fuzzy matching algorithms such as Levenshtein distance, cosine similarity, and locality sensitive hashing to catch data errors in attributes such as names.

Using the process above, a confidence factor can be applied to each match. For those matches where confidence is high, i.e. 95%+, the records can be automatically merged and written to the authoritative single view. Note that the actual confidence factor can vary by use case, and is often dependent on data quality contained in the source systems. For matches below the desired threshold, the merged record with its conflicting attributes can be written to a pending single view record for manual intervention. Inspecting the record to resolve conflicts might be performed by the data steward, or by the application user when they access the record.

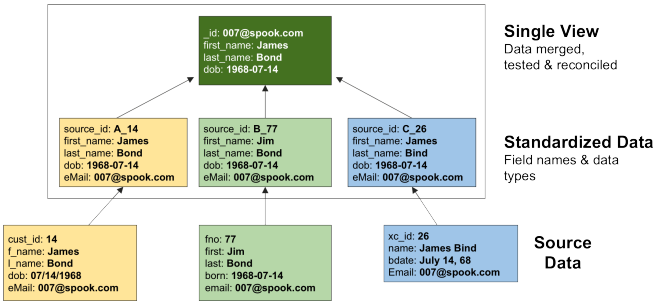

Figure 5: Using MongoDB tools to move from disparate source data to merged and reconciled single view data sets

To assist customers, MongoDB consulting engineers have developed tools to facilitate the process above:

- A Workers framework that parallelizes document-to-document comparisons. The framework allows long running jobs to be partitioned and run over collections of records, maintaining progress of grouping and matching.
- A Grouping tool allows records to be clustered based on attribute similarity, using algorithms such as Levenshtein to calculate the distance between different documents, and then single-linkage clustering to create precise matches for merging.

By combining the Workers framework and Grouping tool, merged master data sets are generated, allowing the project team to begin testing the resulting single view.

Step 8: Architecture Design

While the single view may initially address a subset of users, well-implemented solutions will quickly gain traction across the enterprise. The project team therefore needs to have a well-designed plan for scaling the service and delivering continuous uptime with robust security controls.

MongoDB's Production Readiness consulting engagement will help you achieve just that. Our consulting engineer will collaborate with your devops team to configure MongoDB to satisfy your application's availability, performance, and

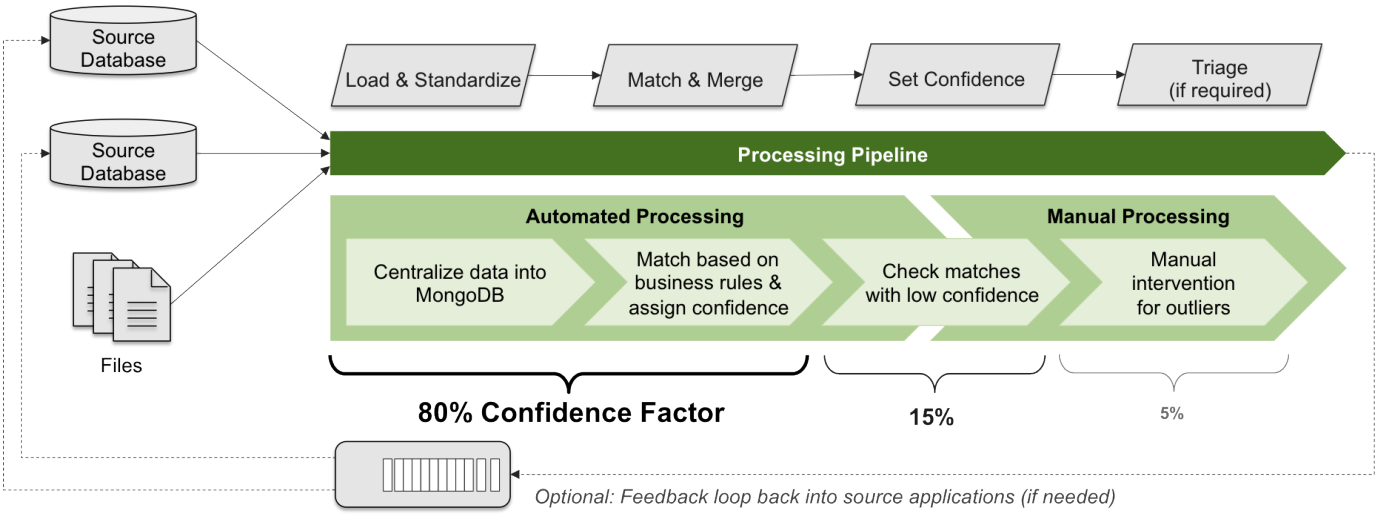

Figure 6: Efficient matching and merging pipeline

security needs, delivering a deployment architecture that will address:

- Where you should place your replica set members for high availability, disaster recovery, and other failover requirements;
- What hardware you should provision to meet performance goals;
- How to enable sharding to scale your single view database as workloads increase;
- How to secure your MongoDB deployment according to your corporate compliance requirements;
- How to continue to meet performance and availability goals as the single view usage evolves and expands.

The MongoDB consulting engineer can then implement these recommendations for you, or your team can handle the required configuration.

Step 9: Modify the Consuming Systems

With the merged data set created and systems provisioned, we can begin modifying the applications that will consume the single view.

The first step will be to create an API that exposes the single view. This will typically be a RESTful web service that abstracts access to the underlying data set. Any number of consuming applications – whether customer-facing web and mobile services, or backend enterprise and analytics applications – can be repointed to the web service, with no or minimal modification the application's underlying logic. Note that write operations will continue to be committed directly to the source systems.

It is generally a best practice to modify one consuming application at a time, thus phasing the development team's effort to ensure correct operation, while minimizing business risk.

Step 10: Implement Maintenance Processes

No organization is static. Digital transformation initiatives supported by agile development methodologies are enabling enterprises to innovate faster – whether through launching new services or evolving existing applications. Our single view data model needs to maintain pace with business change. This change can manifest itself in adding new attributes from existing source systems, onboarding entirely new data sources, or creating new application uses for the single view.

The project team needs to institutionalize governance around these maintenance processes, defining a strategy on how application changes that generate new attributes of value are integrated into the single view schema. The steps defined above – scoping the required changes, identifying the data producers and stewards, updating the schema, and determining the load and merge strategies – are all essential to maintaining the single view. In some

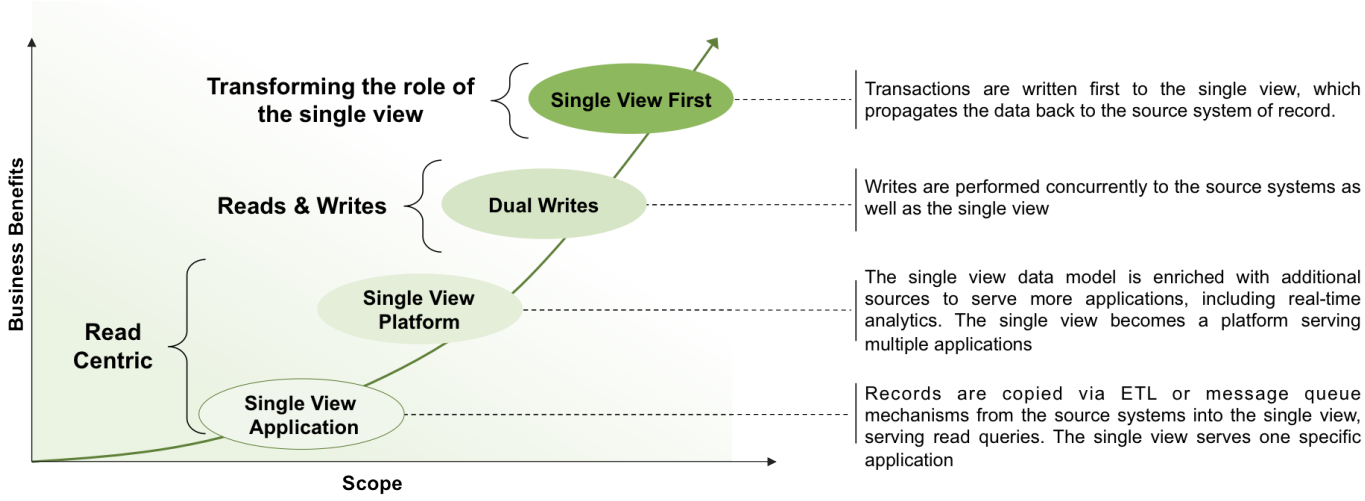

Figure 7: Single view maturity model

more mature single view projects, the application team may decide to write new attributes directly to the single view, thus avoiding the need to update the legacy relational schemas of source systems. This is discussed in more detail in the Maturity Model section of the whitepaper.

As we consider the maintenance process, the benefits of a flexible schema – such as that offered by MongoDB's document data model – cannot be underestimated. As we will see in the Case Studies section later, the rigidity of a traditional relational data model has prevented the single view schema from evolving as source systems are updated. This inflexibility has scuppered many single view projects in the past.

Single View Maturity Model

As discussed above, most single view projects start by offering a read-only view of data aggregated from the source systems. But as projects mature, we have seen customers start to write to the single view. Initially they may start writing simultaneously to the source systems and single view to prove efficacy – before then writing to the single view first, and propagating updates back to the source systems.. The evolution path of single view maturity is shown below.

What are the advantages of writing directly to the single view?

- Real-time view of the data. Users are consuming the freshest version of the data, rather than waiting for updates to propagate from the source systems to the single view.
- Reduced application complexity. Read and write operations no longer need to be segregated between different systems. Of course, it is necessary to then implement a change data capture process that pushes writes against the single view back to the source databases. However, in a well designed system, the mechanism need only be implemented once for all applications, rather than read/write segregation duplicated across the application estate.
- Enhanced application agility. With traditional relational databases running the source systems, it can take weeks or months worth of developer and DBA effort to update schemas to support new application functionality. MongoDB's flexible data model with a dynamic schema makes the addition of new fields a runtime operation, allowing the organization to evolve applications more rapidly.

Figure 8 shows an architectural approach to synchronizing writes against the single view back to the source systems. Writes to the single view are pushed into a dedicated update queue, or directly into an ETL pipeline or message queue. Again, MongoDB consulting engineers can assist with defining the most appropriate architecture.

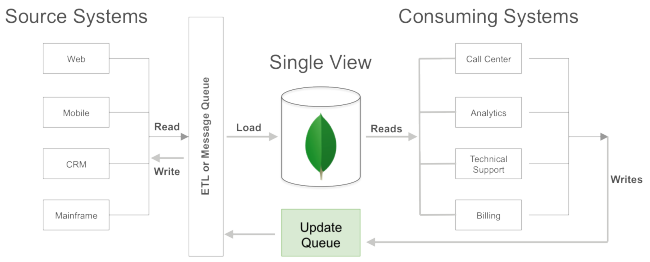

Figure 8: Writing to the single view

Required Database Capabilities to Support the Single View

The database used to store and manage the single view provides the core technology foundation for the project. Selection of the right database to power the single view is critical to determining success or failure.

Relational databases, once the default choice for enterprise applications, are unsuitable for single view use cases. The database is forced to simultaneously accommodate the schema complexity of all source systems, requiring significant upfront schema design effort. Any subsequent changes in any of the source systems' schema – for example, when adding new application functionality – will break the single view schema. The schema must be updated, often causing application downtime. Adding new data sources multiplies the complexity of adapting the relational schema.

Relational databases also struggle to meet the performance SLAs of the system. Typically the single view data set will be normalized across multiple tables, which must then be JOINed to materialize the single view. This process can add significant query latency, while also inhibiting scalability as the single view scales to onboard new data sources and serve new applications.

MongoDB provides a mature, proven alternative to the relational database for enterprise applications, including single view projects. MongoDB is the most popular and widely used non-relational database available today. With its unique Nexus Architecture, MongoDB is at the center of digital transformation initiatives across a range of organizations including ADP, Air France, AstraZeneca, Barclays, Bosch, Cisco, Forbes, KPMG, Lockheed Martin, MetLife, the UK Government's Digital Service, UPS, Verizon, and many more. MongoDB is the only database that harnesses the innovations of NoSQL – data model flexibility, always-on global deployments, and scalability – while maintaining the foundation of rich query capabilities and management integrations that have made relational databases an essential technology for enterprise applications over the past three decades.

As discussed below, the required capabilities demanded by a single view project are well served by MongoDB:

Flexible Data Model

MongoDB's document data model makes it easy for developers to store and combine data of any structure within the database, without giving up sophisticated validation rules to govern data quality. The schema can be dynamically modified without application or database downtime. If, for example, we want to start to store geospatial data associated with a specific customer event, the application simply writes the updated object to the database, without costly schema modifications or redesign.

MongoDB documents are typically modeled to localize all data for a given entity – such as a financial asset class or user – into a single document, rather than spreading it across multiple relational tables. Document access can be completed in a single MongoDB operation, rather than having to JOIN separate tables spread across the database. As a result of this data localization, application performance is often much higher when using MongoDB, which can be the decisive factor in improving customer experience.

Intelligent Insights, Delivered in Real Time

With all relevant data for our business entity consolidated into a single view, it is possible to run sophisticated analytics against it. For example, we can start to analyze customer behavior to better identify cross-sell and upsell opportunities, or risk of churn or fraud. Analytics and machine learning must be able to run across vast swathes of data stored in the single view. Traditional data warehouse technologies are unable to economically store and process these data volumes at scale. Hadoop-based platforms are unable to serve the models generated from this analysis, or perform ad-hoc investigative queries with

the low latency demanded by real-time operational systems.

The MongoDB query language and rich secondary indexes enable developers to build applications that can query and analyze the data in multiple ways. Data can be accessed by single keys, ranges, text search, graph, and geospatial queries through to complex aggregations and MapReduce jobs, returning responses in milliseconds. Data can be dynamically enriched with elements such as user identity, location, and last access time to add context to events, providing behavioral insights and actionable customer intelligence. Complex queries are executed natively in the database without having to use additional analytics frameworks or tools, and avoiding the latency that comes from ETL processes that are necessary to move data between operational and analytical systems in legacy enterprise architectures.

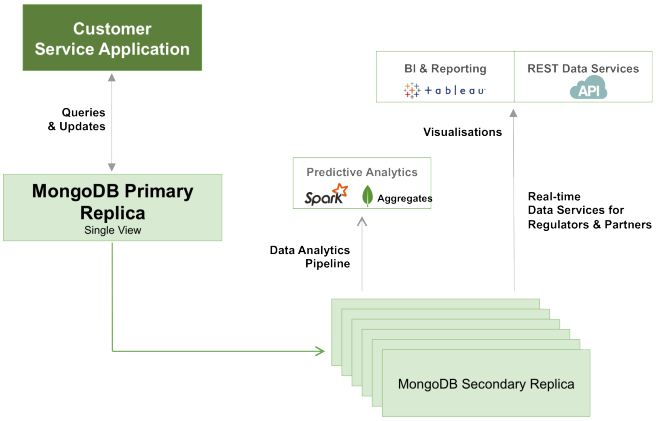

Figure 9: Single view platform serving operational and analytical workloads

MongoDB replica sets can be provisioned with dedicated analytics nodes. This allows data scientists and business analysts to simultaneously run exploratory queries and generate reports and machine learning models against live data, without impacting nodes serving the single view to operational applications, again avoiding lengthy ETL cycles.

Predictable Scalability with Always-on Availability

Successful single view projects tend to become very popular, very quickly. As new data sources and attributes, along with additional consumers such as applications, channels, and users are onboarded, so demands for processing and storage capacity quickly grow.

Figure 10: MongoDB scale-out as the single view grows

To address these demands, MongoDB provides horizontal scale-out for the single view database on low cost, commodity hardware using a technique called sharding, which is transparent to applications. Sharding distributes data across multiple database instances. Sharding allows MongoDB deployments to address the hardware limitations of a single server, such as bottlenecks in CPU, RAM, or storage I/O, without adding complexity to the application. MongoDB automatically balances single view data in the cluster as the data set grows or the size of the cluster increases or decreases.

MongoDB maintains multiple replicas of the data to maintain database availability. Replica failures are self-healing, and so single view applications remain unaffected by underlying system outages or planned maintenance. Replicas can be distributed across regions for disaster recovery and data locality to support global user bases.

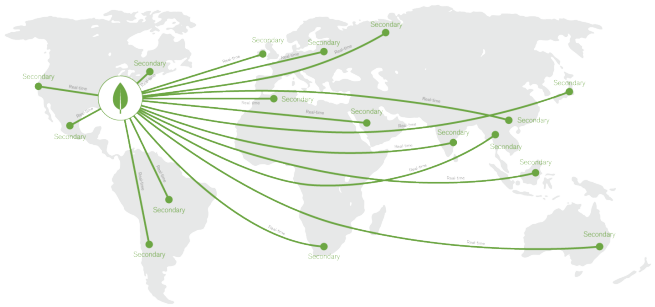

Figure 11: Global distribution of the single view

Enterprise Deployment Model

MongoDB can be run on a variety of platforms – from commodity x86 and ARM-based servers, through to IBM Power and zSeries systems. You can deploy MongoDB onto servers running in your own data center, or public and hybrid clouds. With the MongoDB Atlas service, we can even run the database for you.

MongoDB Enterprise Advanced is the production-certified, secure, and supported version of MongoDB, offering:

- **Advanced Security**. Robust access controls via LDAP, Active Directory, Kerberos, x.509 PKI certificates, and role-based access control to ensure a separation of privileges across applications and users. Data anonymization can be enforced by read-only views to protect sensitive, personally identifiable information. Data in flight and at rest can be encrypted to FIPS 140-2 standards, and an auditing framework for forensic analysis is provided.
- **Automated Deployment and Upgrades**. With Ops Manager, operations teams can deploy and upgrade distributed MongoDB clusters in seconds, using a powerful GUI or programmatic API.
- **Point-in-time Recovery**. Continuous backup and consistent snapshots of distributed clusters allow seamless data recovery in the event of system failures or application errors.

Single View in Action

MongoDB has been used in many single view projects. The following case studies highlight several examples.

MetLife: From Stalled to Success in 3 Months

In 2011, MetLife's new executive team knew they had to transform how the insurance giant catered to customers. The business wanted to harness data to create a 360-degree view of its customers so it could know and talk to each of its more than 100 million clients as individuals. But the Fortune 50 company had already spent many years trying unsuccessfully to develop this kind of centralized system using relational databases.

Which is why the 150-year old insurer turned to MongoDB. Using MongoDB's technology over just 2 weeks, MetLife created a working prototype of a new system that pulled together every single relevant piece of customer information about each client. Three months later, the finished version of this new system, called the 'MetLife Wall,' was in production across MetLife's call centers.

The Wall collects vast amounts of structured and unstructured information from MetLife's more than 70 different administrative systems. After many years of trying, MetLife solved one of the biggest data challenges dogging companies today. All by using MongoDB's innovative approach for organizing massive amounts of data. You can learn more from the case study.

CERN: Delivering a Single View of Data from the LHC to Accelerate Scientific Research and Discovery

The European Organisation for Nuclear Research, known as CERN, plays a leading role in the fundamental studies of physics. It has been instrumental in many key global innovations and breakthroughs, and today operates the world's largest particle physics laboratory. The Large Hadron Collider (LHC) nestled under the mountains on the Swiss - Franco border is central to its research into origins of the universe.

Using MongoDB, CERN built a multi-data center Data Aggregation System accessed by over 3,000 physicists from nearly 200 research institutions across the globe. MongoDB provides the ability for researchers to search and aggregate information distributed across all of the backend data services, and bring that data into a single view.

MongoDB was selected for the project based on its flexible schema, providing the ability to ingest and store data of any structure. In addition, its rich query language and extensive secondary indexes gives users fast and flexible access to data by any query pattern. This can range from simple key-value look-ups, through to complex search, traversals and aggregations across rich data structures, including embedded sub-documents and arrays.

You can learn more from the case study.

Global Airline: Enabling Customer Intimacy

"Big data" is at the core of the airline's digital transformation journey, touching many areas of the business – from aircraft maintenance, to cargo and operations, to corporate HR and finance functions. It is in

the domain of customer experience that the airline has prioritized its initial investments.

The company wanted to optimize the customer journey across web, mobile, call center, and partner channels. However, with customer data spread across many different source systems, it was impossible to unify and personalize the customer experience.

To address these challenges, the airline has built a single customer view platform on MongoDB. Data is ingested from data sources using Apache Kafka and then processed with Apache Spark, before then being written to the single view stored in MongoDB. The single view serves real-time customer interactions from multiple channels, enabling a consistent, personalized experienced, achieving heightened levels of customer service and loyalty.

Conclusion

Bringing together disparate data into a single view is a challenging undertaking. However, by partnering with a vendor that combines proven methodologies, tools, and technologies, organizations can innovate faster, with lower risk and cost. MongoDB is that vendor.

We Can Help

We are the MongoDB experts. Over 2,000 organizations rely on our commercial products, including startups and more than half of the Fortune 100. We offer software and services to make your life easier:

MongoDB Enterprise Advanced is the best way to run MongoDB in your data center. It's a finely-tuned package of advanced software, support, certifications, and other services designed for the way you do business.

MongoDB Atlas is a database as a service for MongoDB, letting you focus on apps instead of ops. With MongoDB Atlas, you only pay for what you use with a convenient hourly billing model. With the click of a button, you can scale up and down when you need to, with no downtime, full security, and high performance.

MongoDB Cloud Manager is a cloud-based tool that helps you manage MongoDB on your own infrastructure. With automated provisioning, fine-grained monitoring, and continuous backups, you get a full management suite that reduces operational overhead, while maintaining full control over your databases.

MongoDB Professional helps you manage your deployment and keep it running smoothly. It includes support from MongoDB engineers, as well as access to MongoDB Cloud Manager.

Development Support helps you get up and running quickly. It gives you a complete package of software and services for the early stages of your project.

MongoDB Consulting packages get you to production faster, help you tune performance in production, help you scale, and free you up to focus on your next release.

MongoDB Training helps you become a MongoDB expert, from design to operating mission-critical systems at scale. Whether you're a developer, DBA, or architect, we can make you better at MongoDB.

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 Enterprise Download (mongodb.com/download)
MongoDB Atlas database as a service for MongoDB (mongodb.com/cloud)

New York • Palo Alto • Washington, D.C. • London • Dublin • Barcelona • Sydney • Tel Aviv
US 866-237-8815 • INTL +1-650-440-4474 • info@mongodb.com