TOWARD A MODERN DATA ARCHITECTURE FOR EMBEDDED ANALYTICS

Data has gone from a mere byproduct of applications and processes to being crowned the king of the modern world. The Economist went so far as to declare data, not oil, as the world's most valuable resource[1]. The data landscape is also more complex than oil's—with abundant sources, volume, velocity, and variety.

Given data's high demand and complex landscape, data architecture has become increasingly important for organizations that are embarking on any data-driven project—including embedded analytics. Before you embed new analytics, dashboards, or reporting capabilities in your software, you need to carefully design a data architecture with analytics in mind. Why? Because a poorly conceived data strategy can negatively impact the response time and performance of your entire application.

Unfortunately, roadmaps are urgent and priorities are scattered. It's tempting to overlook or postpone the data architecture steps of your embedded analytics project. Some embedded analytics vendors may even mislead you, saying you don't need to worry about data architecture because their platforms integrate well across all environments. And the fact is, your application development team may not fully understand how embedded analytics will impact the production environment and performance.

Inevitably, any modern data project will reach a point where performance and data complexity force you to re-evaluate the data architecture.

This white paper discusses data analytics architecture in four parts:

1 *The Economist: "The world's most valuable resource is no longer oil, but data,"* May 2017

Part 1: Considerations when planning your embedded data analytics architecture

The road to an ideal data architecture is long and rarely straight. Don't expect to revamp your entire data architecture instantaneously. Your data journey will depend on your business priorities, your timeline, and the needs of your customers.

Ask yourself the following questions to help inform your architecture plan:

What are your data analytics goals? Your data architecture will hinge on your goals. Are you generating reports to describe "what happened" (descriptive analytics), or giving people insight into what will happen based on the past (predictive analytics)? Or are you automating decisions based on patterns in data (prescriptive analytics)? Moving from descriptive to prescriptive analytics requires a progressively complex data architecture.

Who is your end user? Your data analytics goals will be tied closely to your end customer. Are your users simply trying to find information within the context of your application (for example, report writers gathering information from a portal)? Is your end user a data analyst who needs to understand performance and data structure? Is he a data scientist who wants to bring in datasets from different sources and manipulate them within your application? If you have multiple end users with different personas, you may need a data architecture strategy that accommodates scaling and progressive complexity.

What types of content will be delivered and how? Understanding what type of content will be delivered in your embedded analytics tool and how it will be delivered is an important part of your strategy. For example, a tool that runs and emails reports has different performance characteristics and requirements than an interactive dashboard for an executive or a self-service user.

What data structures do you have in place? How your data is structured (e.g., array, file, record, table, tree) will determine how that data is stored and organized, and more importantly, how it can be accessed.

What are your data latency expectations? Data freshness is a metric that measures how quickly data updates are collected, processed, and made available in analytics reports. Do your end users demand access to real-time and near-time operational data? A well-designed data architecture can help you reduce data latency. Also consider how frequent your data updates are.

Where are you in your project timeline? Are you just starting out in your embedded analytics journey, or are you looking to optimize what you have in place? Are you looking for quick wins with plans to improve your analytics offering based on customer feedback, or do you have a longer timeline to integrate an analytics workflow in your application with more robust features?

How do you plan to scale your current application infrastructure and platform? Are you planning to scale up with new upgrades or scale out by linking to other resources? Growing data volumes will affect your performance, so data size matters. It is therefore important to design your data architecture with scalability in mind.

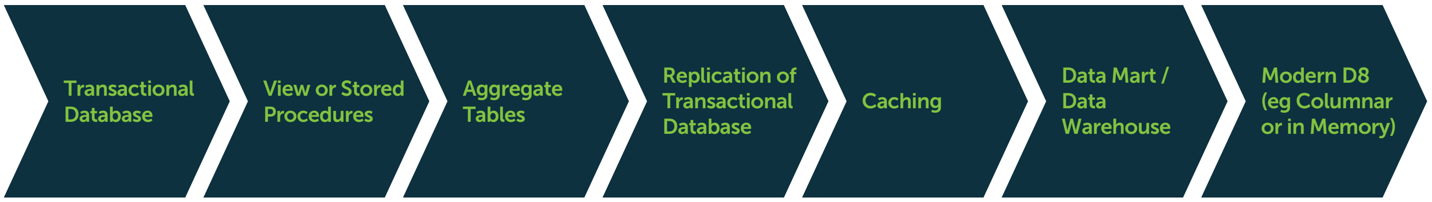
What skills, tools, and budget does your team have? Does your project team have the skillset to manage data issues? What tools are used to manage your data architecture? How do you monitor the different layers to detect performance issues? If you don't have the skillset or tools, does your budget allow you to consult with embedded analytics vendors who can conduct performance reviews and provide advice and troubleshooting?

How will you iterate over time? Most embedded analytics projects are iterative, and will change as your users demand new capabilities and new technologies emerge. Your data architecture will evolve with the different iterations of your embedded dashboards and reports.

Part 2: Seven Common Approaches to Data Architecture for Embedded Analytics and Recommendations

Depending on your organization's particular setup and needs, you could take a number of different approaches to data architecture for embedded analytics. Here, we outline the seven most common approaches—from a transactional database to a modern analytics database (columnar or in memory). Note that these are not individual steps in your data architecture journey. You may skip some approaches altogether, or use two simultaneously. This is not a prescribed path.

Read on to explore each step, the pros and cons, and the impact on analytic workloads, as well as our recommendations.


Transactional Database
View or Stored Procedures
Aggregate Tables
Replication of Transactional Database
Caching
Data Mart / Data Warehouse
Modern D8 (eg Columnar or in Memory)

Transactional Database

The starting point for many application development teams is the ubiquitous transactional database, which runs most production systems. Although they aren't specifically built for analytics, transactional databases typically become the default analytics environment because they are already in place, and are familiar and accessible.

Transactional databases are row stores, with each record/row keeping relevant information together. For example, for a travel website, each record/row may store relevant customer information, flight number, date of booking, and so on, and may be uniquely identified by Customer_ID. The database may contain several tables tracking other information related to the transaction, such as flight schedule, ticket prices, and country list. Transactional databases are known for very fast read/write updates and high data integrity.

The pros of transactional databases include:

- Most application development teams are familiar with this database structure and understand how to write queries to get the correct data.
- As soon as data hits the transactional database, it is available for analytics.

The main downside of transactional databases is structure: They're not designed for optimal analytics queries. This creates the following issues:

- You have to resort to complex joins and operations to gain insight into the data stored in a transactional database. In our travel website example, if you want to correlate the most popular flight bookings with country and time of year for a marketing campaign, you would have to come up with unwieldy joins across several tables.
- Typically, performance of analytic queries will affect the performance of the transactional database —and that may create lags in response time with both the analytics query and transactional system.

Bottom Line: Using transactional databases for embedded analytics makes sense if you already have them in place, but you will eventually run into limitations and need workarounds.

View or Stored Procedures

Typically, when developers start noticing problems with their transactional systems, they may opt to create some views or stored procedures. Views create the appearance of a table as a result set of a stored query. The view could be generated from a combination of rows and columns across multiple tables. While views only showcase the data, stored procedures allow you to execute SQL statements on the data.

The pros of using views and stored procedures include:

- Simplifies the SQL needed to run analytics
- Allows users to filter the information they want to see
- With stored procedures, users can make modifications to the underlying tables (with views, you can filter and sort only).
- Provides a good abstraction layer if schemas are similar in different underlying tables/columns

Views or stored procedures may be the next logical step from transactional databases, but they typically make performance worse. The cons include:

- Complex views with lots of tables create additional joins.
- Stored procedures may result in filtering and post processing after the data is retrieved, which may significantly reduce your response time.

Bottom Line: When it comes to embedded analytics, views or stored procedures risk creating lags and affecting your application's response time.

Aggregate Tables or Material Views

Application development teams may opt to create aggregate tables or material views as another work-around to using view or stored procedures. With an aggregate table, you can create a summary table of the data you need by running a "Group By" SQL query. For example, a marketing department can create an aggregate table that shows "Sales over a month." Since most analytics queries typically involve aggregation, using an aggregate table will prevent the need to aggregate the data for every query. In a materialized view, you can store query results in a table or database.

The pros of using aggregate tables or material views include:

- Simplifies the SQL needed to run analytics
- Aggregating data improves query performance
- No need to aggregate data for every query

The downside of aggregate tables or material views is:

- You need to figure out when and how to update the tables:
- Triggers can help with synchronization of data, but typically add to load on transactional additions/updates and can be complex to implement.
- Update processes, which run on a periodic basis, may be another alternative—but you will need to implement logic that can distinguish between updates versus new transactions.
- Another issue arises if you have new requirements where the necessary data is not in the aggregate tables or material views. You may have to go back to the transactional database to recreate your aggregate tables and start over.

Bottom Line: Pre-aggregated tables and materialized views will help with performance. However, you do need to stay organized and put strict processes in place to keep the aggregates up to date.

Replication of Transactional Database

Replication is another common workaround that offloads analytics queries from the production database to a replicated copy of the database. Replication requires copying and storing data in more than one site or node, so all of the analytics users share the same information.

The pros of replication include:

- Many databases have built-in replication facilities, so this is easier to implement. You can leverage transaction logging to publish changes to replicated databases, then re-apply the transactions.
- Removes analytical load from the production database.
- Allows for different indexing strategy that may create better indexes for analytics to be introduced.
- Aggregate tables can now be created on the replicated database rather than the production database.

  Allows your team to use a familiar database technology.

The main issue with replication is the lag between a new transaction hitting the database and that data being available in the replicated table. Other downsides include:

- The table structures are typically the same and subject to complex queries, creating aggregate tables, etc.
- If database replication is difficult with provided tools, a replication technology may need to be introduced, implemented, and maintained.
- Real-time queries may still need to hit the production database (dependent upon “lag” or data latency expectations).

Bottom Line: Replicating the production database also means replicating the complexity of queries in your embedded analytics solution.

Caching

Sometimes, application development teams may turn to caching to help with query performance. With caching, you can preprocess complex and slow-running queries so the resulting data is easier to access when the user requests the information. The cached location could be in memory, another table in the database, or a file-based system where the resulting data is stored temporarily to make it easier to access frequently.

The pros of caching include:

- Caching can help with performance where queries are repeated.
- Caching can also enable better performance with federated data (data in multiple-source systems), particularly when data needs to be joined or filtered based on other queries.
- It’s relatively easy to set up, in most environments.
- With caching, you query the database once and re-use the data many times.

Caching works well in a stable environment, where data in the database does not change frequently. You will need to identify data to be cached and develop processes to refresh the cache on a periodic basis. Some downsides include:

- If the cache isn't reused, it can add to performance overhead.
- There are data latency issues involved with using caches, and in some cases (like real-time queries), caching may not be the best option.
- Where there are multiple sources, ensuring consistency and scheduling of cache refreshes can be complex.

Bottom Line: Caching can be a quick fix for improving embedded analytics performance, but the complexity of multiple sources and data latency issues may lead to limitations over time.

Data Mart/Data Warehouse

For a more sophisticated data architecture, application development teams may turn to data warehouses or data marts. Data warehouses are central repositories of integrated data from one or more disparate sources. Data marts contain a subset of a data warehouse designed for a specific reason (e.g., isolating data related to a particular line of business within the company).

Data warehouses and data marts allow you to organize your data in a way that simplifies query complexity. Star and snowflake schemas are the common schema models associated with data warehouses and are logically structured to make analytics faster and easier to access.

Other benefits of using a data warehouse or data mart include:

- Query performance is significantly improved. Since all the required data is in the data warehouse, this removes the need for querying multiple sources.
- You reduce the load on the transactional database. You can build a data warehouse and data mart using your existing database technology.

The main downside is that designing a data structure for particular use cases can be complex at first. If you are not familiar with star or snowflake schema and the ETL tools involved, then you may need support through this stage. Other downsides include:

- ETL involves having the skill set to understand and use data transformation tools.
- A need to understand how to lower data latency in scheduling updates to a data warehouse.
- Additional quality control steps are needed to ensure the data in the data warehouse is valid; the new queries are running correctly; and each step in the ETL/ELT process is validated.
- It takes time to build, test, and manage.
- You may still need to occasionally query the transaction system for real-time requirements.

Bottom Line: Data warehouses and data marts are designed for faster analytics and response times, but implementation will take more time and be more complex.

Modern Analytics Database

Even with a data warehouse in place, application development teams find that switching to a modern analytics database may be the best option for analytics queries.

Modern analytics databases are typically columnar structures or in-memory structures. In columnar structures, data is stored at a granular column level in the form of many files. This structure makes it faster to query since only the columns associated with the query need to be read, not the entire row. This significantly increases performance. For in-memory structures, the data is loaded into the memory, which makes reading/writing dramatically faster than a disk-based structure.

Pros of a modern analytics database include:

- Improved performance on data load. ELT is faster, since data transformations occur on the analytics database platform.
- Optimal query performance since the database enables easier management of “flat tables” associated with star schemas.
- Databases that are designed for fast queries and optimized data storage, which is important for large volumes of data.

There is a learning curve associated with switching to a modern analytics database. You will need to learn to operate and support a new database technology. You will also need to learn how to optimize the database performance with new concepts such as projections rather than indexes. Other downsides include:

- Unlike transactional databases, analytics databases perform updates and deletions poorly.
- You may need to implement workarounds for this, such as partition design to improve where data needs updating, or new data transformation strategies to avoid update/deletion issues.
- You also need to determine the update frequency for data.

Bottom Line: The modern analytics database is optimal for faster queries and dealing with large volumes of data, but it requires specialized skills and can be costly to implement.

Part 3: What data approach is right for you?

Some data architecture approaches will be better suited to your organization than others, and you may skip some approaches altogether. It all depends on your particular setup and needs. So how do you find the approach that's best for you?

Consider these common data scenarios and recommended architecture approaches:

- Where is your solution located? On premise or SaaS?
- If your solution is on your customer's premises, then creating Views, Stored procedures and/or Aggregate tables will be the easiest first step to implement.
- If you are hosting the solution for your customers, and performance of your transactional database is critical, then Replication is a good first step. Replication will remove the analytics load from the transactional database. You may add Aggregate tables as a next step and eventually consider moving into a Data Warehouse/Data Mart.

How large and complex is your environment?

- When customers use your solution, does it mean large data volumes where performance is critical? Then investing in Replication and/or a Data Warehouse will justify the additional hardware and costs of implementation, since they will quickly run into the limitations of working with Views and Stored Procedures.

Is analytics a strategic goal in your organization?

- If you host your environment and realize analytics is a strategic goal, then it's worth investing in a Data Warehouse/Data Mart. To design a Data Warehouse/Data Mart effectively, the first step is to make sure you understand the types of queries and use cases for your end user. This allows you to build and implement the environment correctly and optimize it for the types of queries and use cases it will handle.

How large is your Data Warehouse/Data Mart?

- If your starting point is a Data Warehouse/Data Mart, moving to a Modern Analytics Database will depend on how large your current environment is and how critical performance is to your organization.
- Also, if you are hosting your own Data Warehouse/Data Mart, you can save money in the long term with less hardware overall.

Ultimately, your current technology and environment, skills, and performance needs will be critical drivers in your data architecture decision.

Now that we have examined how your analytics data architecture may evolve in practice, let's take a look at what industry analysts recommend for an ideal modern data architecture.

Part 4: Toward a modern data analytics architecture

Industry analysts have more to say about choosing a data architecture approach. In the Eckerson Group report, *A Pragmatic Approach to Modern Data Architecture*, David Wells explains the first step to modernizing your data architecture is to change your perspective on data. He writes, "Think of data not as something that is static and stored, but as something live, dynamic, and flowing through every business process."

What does this mean for the analytics architecture? It's a shift away from traditional BI architecture—which is based on a linear data flow, structured data, rigid infrastructure, batch processing, and centralized services—and toward a modern approach based on multi-directional data flow, iterative workflow, both structured and unstructured data, elastic infrastructure, and decentralized/self-service services. Gartner's *Planning Guide for Data and Analytics* goes into more detail on conceiving an end-to-end analytics architecture that fuses data, insight, and action. In this architecture, there are four phases: data acquisition, organization, analysis, and delivery. The table on the next page summarizes the main guidelines related to each of these phases.

Table 1-1: Modern data architecture guidelines

Phase	Guidelines	Description
Data Acquisition	**Expand data sources**	Create a flow of incoming data that incorporates cloud and external streaming data (such as geolocation, weather, consumer data, etc.). The idea is to shift from just "storing" data and hoping someone will use it to thinking about the information that will help downstream processes and people.
	Use machine learning at source	Deploy machine learning algorithms to assess incoming data and decide whether, when, and how the data will be stored/used.
Organization	**Logical data warehouse**	Instead of storing data in a monolithic warehouse, which can quickly go stale, organizations can deploy a logical data warehouse (LDW) to dynamically connect relevant data across heterogeneous platforms.
	Use virtualization tools for data layer	Gartner recommends the use of virtualization tools to create a data virtualization layer on top of the LDW. This establishes a shared data access layer, regardless of source, and logically relates the data.
Analysis	**Support a range of analytics capabilities**	Incorporate a range of analytics capabilities (from descriptive to prescriptive and predictive analytics) within the workflow of the organization. Let the data guide interactions and analytics, and use algorithms to drive processes.

Choosing the right approach to data architecture will depend on your organization's particular needs. While it may seem overwhelming to modernize your data analytics architecture—especially if developing embedded analytics is not a core competency of your engineering and product teams—it can be done.

Gartner's 2019 Planning Guide for Data and Analytics advises application development teams to start small and build in stages, beginning with a business challenge that can be solved. You may also consider partnering with a third-party embedded analytics platform that is already built on modern data architecture principles. This will allow you to scale your embedded analytics as your application continues to grow and evolve.

Learn more about how embedded analytics fits in your tech stack.
Read the ebook: Are Your Embedded Analytics DevOps-Friendly?