DATA VIRTUALITY MASTERCLASS

Topic: Implementing DV Best Practices

What to expect from this session?

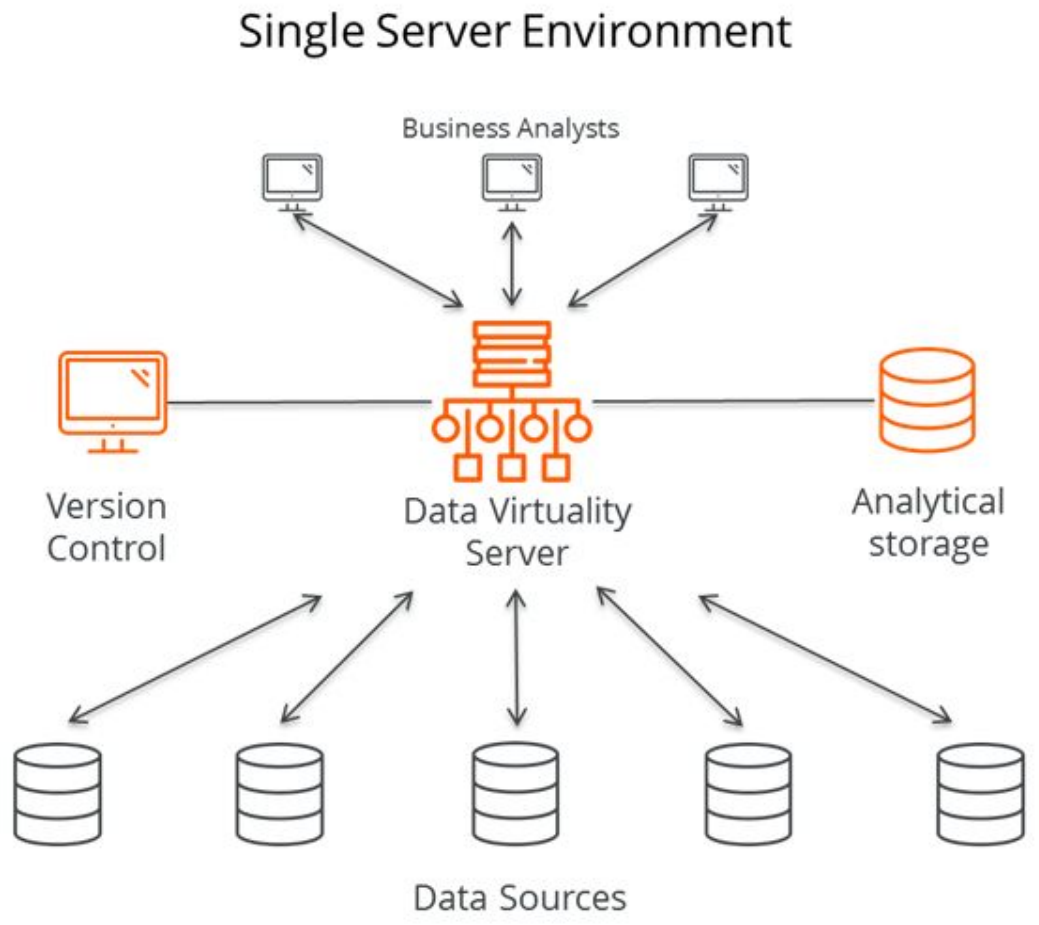
This track will tell you how to apply the Data Virtuality Best Practices:

- Server Environments - Single vs Multiple
- Version Control
- Data Source Onboarding
- Structuring the view hierarchy with a layered approach
- Naming conventions
- Materialization considerations
- LDW Sizing

Server Environments - Single vs Multiple

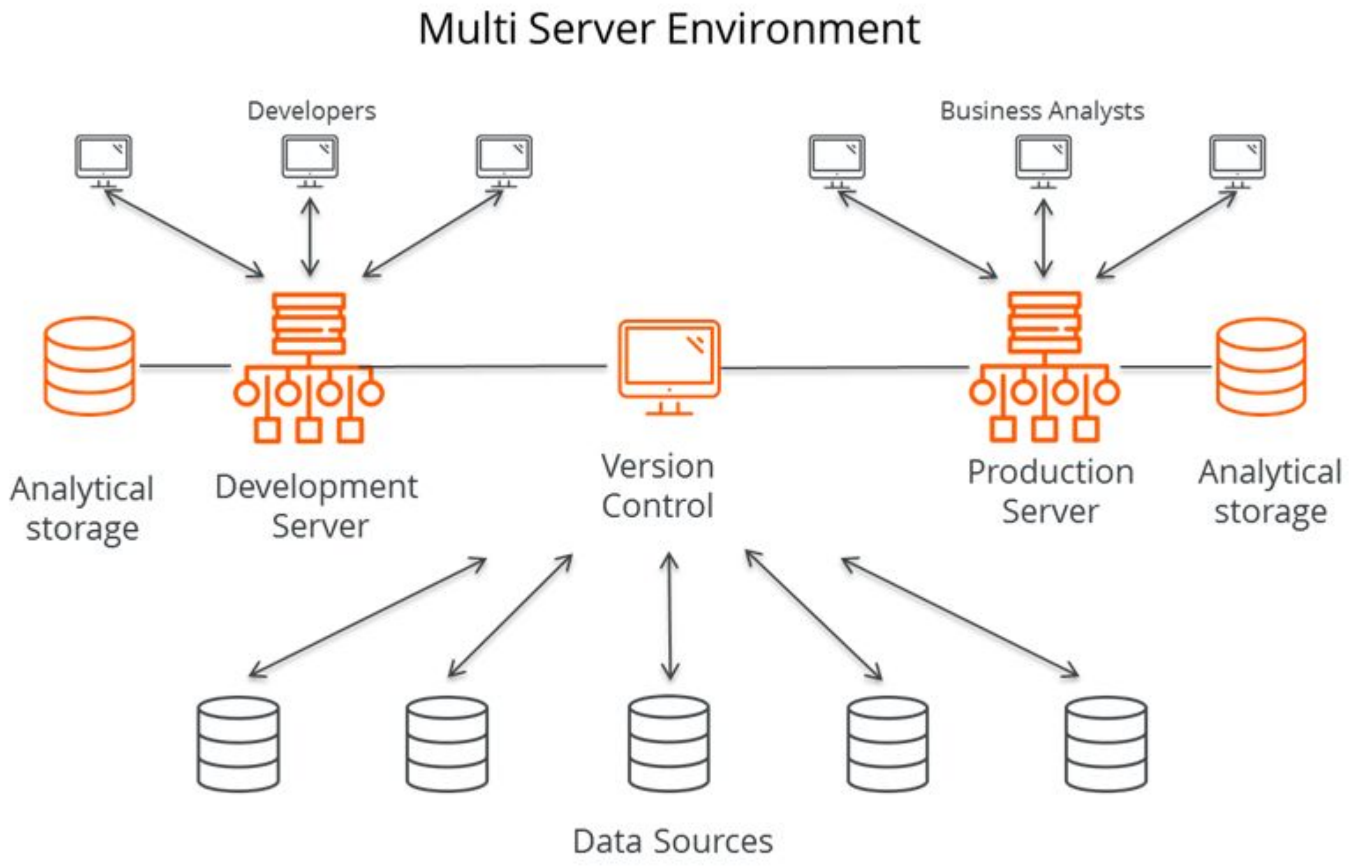
Server Environments - Single Server

- Quick and easy setup
- No synchronization
- Challenge to test new developments
- Risk for changes
- Requires strong backup strategy



Server environments: Multiple Servers

- Scenarios:
    - Dev - Prod
    - Dev - UAT - Prod
- Naming: _Prod or _Dev
- Important: Analytical storage for each
    - because of mat_tables
- DV Sync takes care to keep them synced
    - In Enterprise scenarios, this can be a challenge

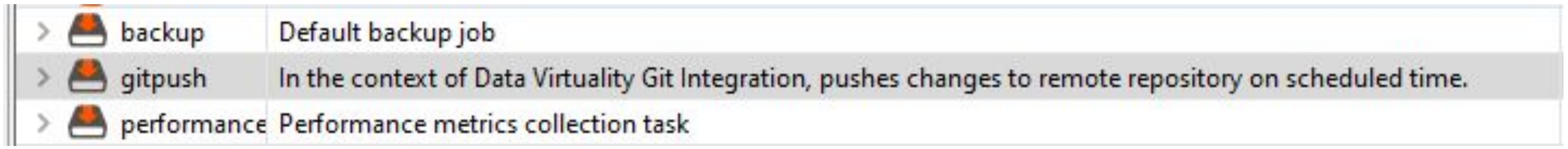


Demo - DV Sync

Version control

Version Control

- Export job is standard
- Own exports can be created and automated via ../dvserver/bin/cli-export-1.0/..
- Changes of objects are written in the SYSLOG schema
  - but cleaned away, make sure to batch replicate it
- The recommended approach was to check in the exported SQL
- Git integration is reaching the product:



backup	Default backup job
gitpush	In the context of Data Virtuality Git Integration, pushes changes to remote repository on scheduled time.
performance	Performance metrics collection task

Version Control - advanced Export Tool settings

5) Export using create* statements instead of import*:

```
export.bat --username admin --password admin --host localhost --ds-validation true --opt-validation true --export-mattable-state false
--view-proc-validation true
```

6) Export with drop/delete statements before creation:

```
export.bat --username admin --password admin --host localhost --purge-system-data true
```

8) Create separate files for export of data sources, virtual schemas with views and procedures and the rest:

```
export.bat --username admin --password admin --host localhost --use-model-file true
```

9) Export JBoss settings as well

```
export.bat --username admin --password admin --host localhost --export-jboss-settings true
```

Demo - Export Tool

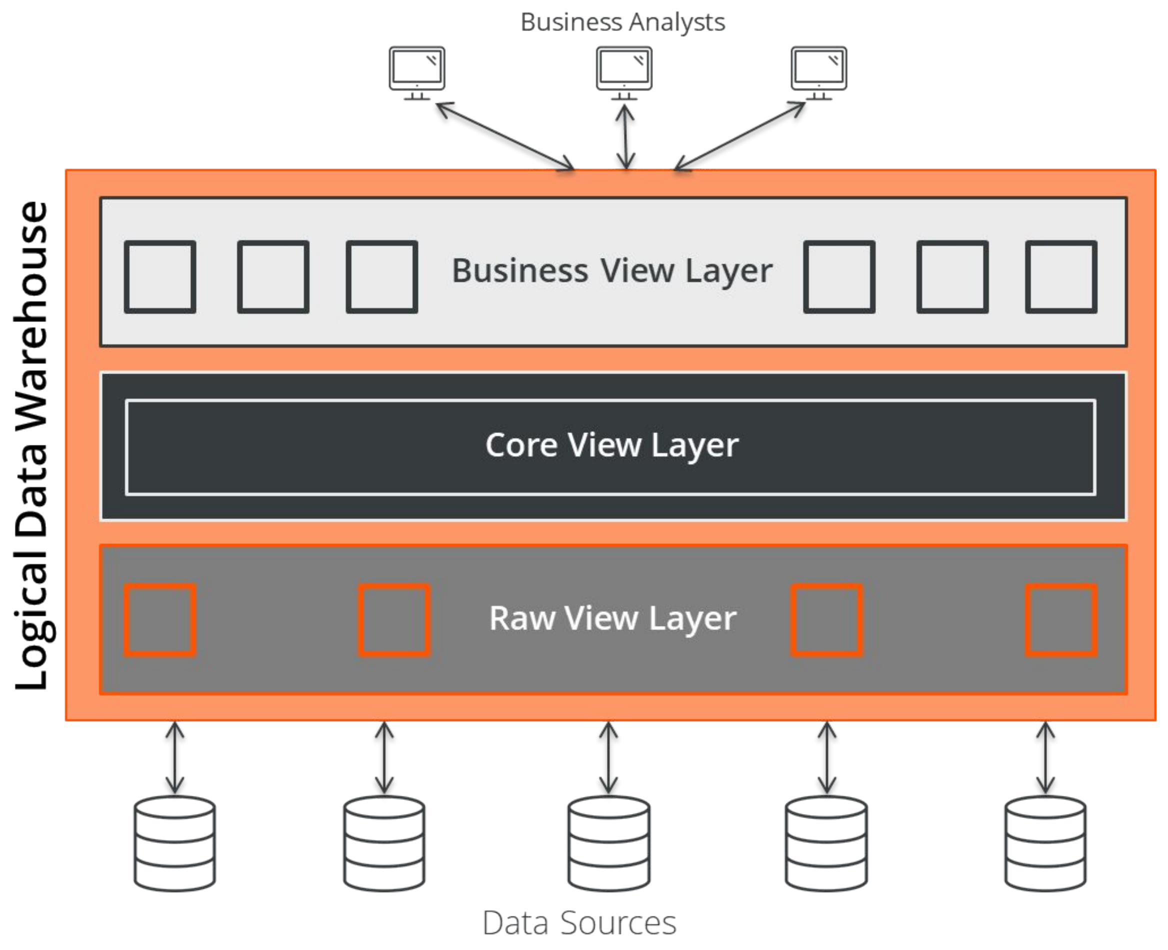
Data Source onboarding

Data Source onboarding

- Here we are looking at databases
- Goals:
  - Analytical access
  - Stability and performance - when talking about a production data source
- Put raw views on top and control permissions there
  - It is easier to provide permissions at this level, rather than having to control multiple access points
- Other considerations to discuss:
  - geographic distance
  - source system performance
  - type and schema of data
  - expected frequency of use
- Options for critical data sources: secondary copy and CDC

Structuring the view hierarchy with a layered approach

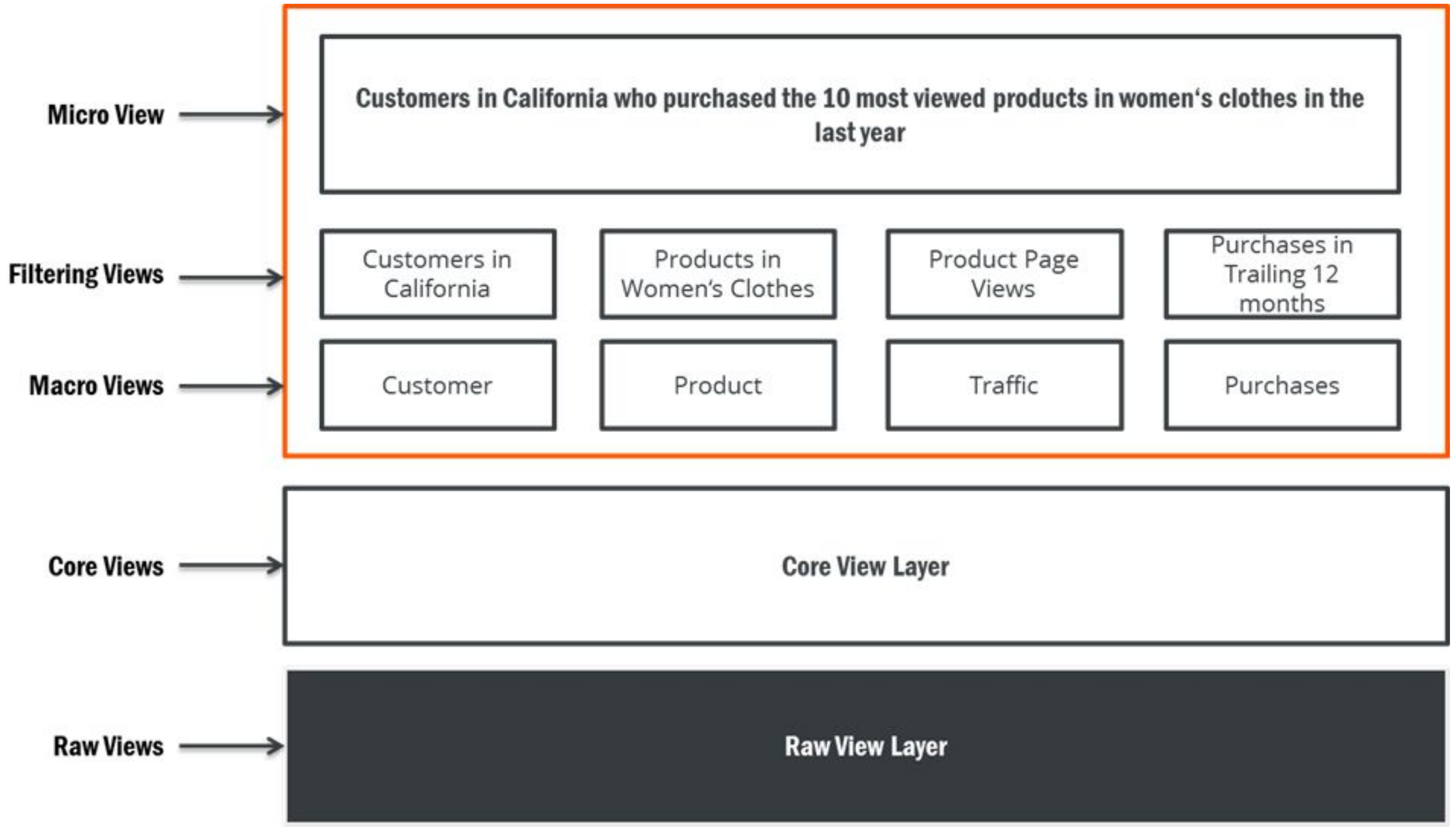
Logical Data Warehouse Schema



Naming conventions

Object Type	Purpose	Example Naming Convention
Data Source	The original data sources existing outside of your Logical Data Warehouse. Examples: Oracle, Financial systems, CRM, etc.	Prepend the data source name with 'DataSource_'. Examples: DataSource_Oracle DataSource_Fin DataSource_CRM
Raw Views	Raw views represent tables from source data systems that appear almost precisely as they do in the source system. Raw views are stored in schemas named to represent the original data source. So for example if we onboard our enterprise data warehouse named 'EDW' we will create a schema named Raw_EDW and we will store all raw views from that source in this schema. The raw views in this schema will each represent a table from the EDW in this example. Each raw view represents a table in a source system looking just as it does in the source system with minimal renaming, transformations, or cleansing. This approach allows quick and easy schema updates in the event that the source system schema changes.	Prepend the raw view schemas with 'Raw_' Examples: Raw_Oracle Raw_EDW
Core Views	All core views are stored in a single core view schema that could be named Core_Views for easy identification. A core view is a view into a source system in which filtering, naming changes, and modeling would be applied prior to any virtual joins occurring. An example change made in the core layer might be to present the customer name in a core view in a field identified as "Customer_Name" whereas in the source system it may be in multiple fields and the field names may be more cryptic such as EDW_CUS_FIR, EDW_CUS_LAS, etc. The business friendly semantic name makes browsing and discovering data much easier since it removes the guesswork from field identification. In addition, the entire table can be renamed at this layer to further clarify stored data.	Place all core views in a schema named "Core_Views"
Business Views	Business views are virtualized views representing sets of data joined across one or more data sources. Business views are stored in virtual schemas that are named to represent the business unit or business function. Business views are views joining one or more tables from disparate data sources. Semantic naming, aggregate functions, and filtering can all be applied in business views to make them business-friendly and to ensure they are as useful as possible.	Prepend business view schemas with 'BusinessViews_' Examples: BusinessViews_Customer_360View BusinessViews_Finance

Virtual Data Building Blocks


Micro View
Filtering Views
Macro Views
Core Views
Raw Views
Customers in California who purchased the 10 most viewed products in women's clothes in the last year
Customers in California
Products in Women's Clothes
Product Page Views
Purchases in Trailing 12 months
Customer
Product
Traffic
Purchases
Core View Layer
Raw View Layer

Access Roles

Role	Typical Member	Development Environment Rights	Production Environment Rights
DV Admin	Operational Support Technology Platform 'Owner'	Server configuration Install upgrades and patches Onboard data sources Create shared virtual views Create personal virtual views Modify existing views of others Modify own existing views Setup and configure caching/materialization Connect to & query live data Connect to & query materialized data	Server configuration Install upgrades and patches Onboard data sources Create shared virtual views Create personal virtual views Modify existing views of others Modify own existing views Setup and configure caching/materialization Connect to & query live data Connect to & query materialized data
DV Power User	Business Group Analyst Lead	Onboard data sources Create shared virtual views Create personal virtual views Modify existing views of others Modify own existing views Setup and configure caching/materialization Connect to & query live data Connect to & query materialized data	Create shared virtual views Create personal virtual views Modify own existing views Connect to & query live data Connect to & query materialized data
DV User	Business Analyst	Create personal virtual views Modify own existing views Setup and configure caching/materialization Connect to & query live data Connect to & query materialized data	Create personal virtual views Modify own existing views Connect to & query materialized data

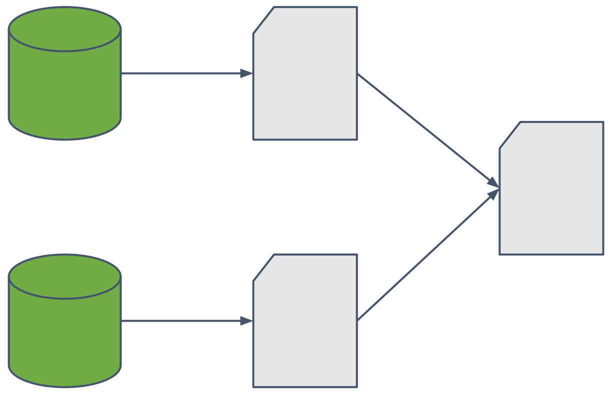
Materialization considerations

Materialization Considerations - Motivation

- Why materialize?
  - Protect the data source (operational system) from performance impacts
  - Improve performance for queries against raw and aggregated data
  - Leverage Analytical Storage MPP capabilities (if you join materialized data, it will be an operation on the analytical storage)

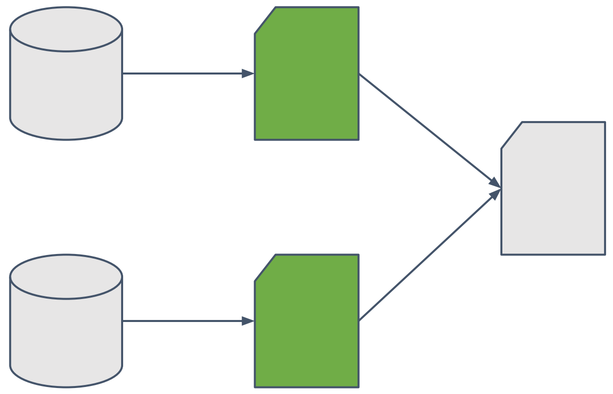
Materialization Considerations - Where to materialize

- Where to materialize in the hierarchy of data sources and views?
  - Depends on your requirements - performance vs flexibility



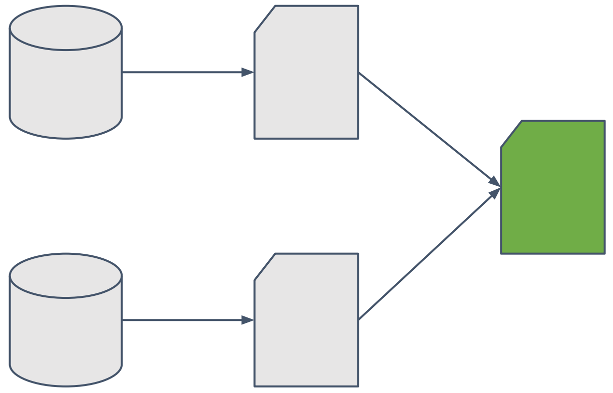
Lowest performance

Highest flexibility



Medium performance

Medium flexibility

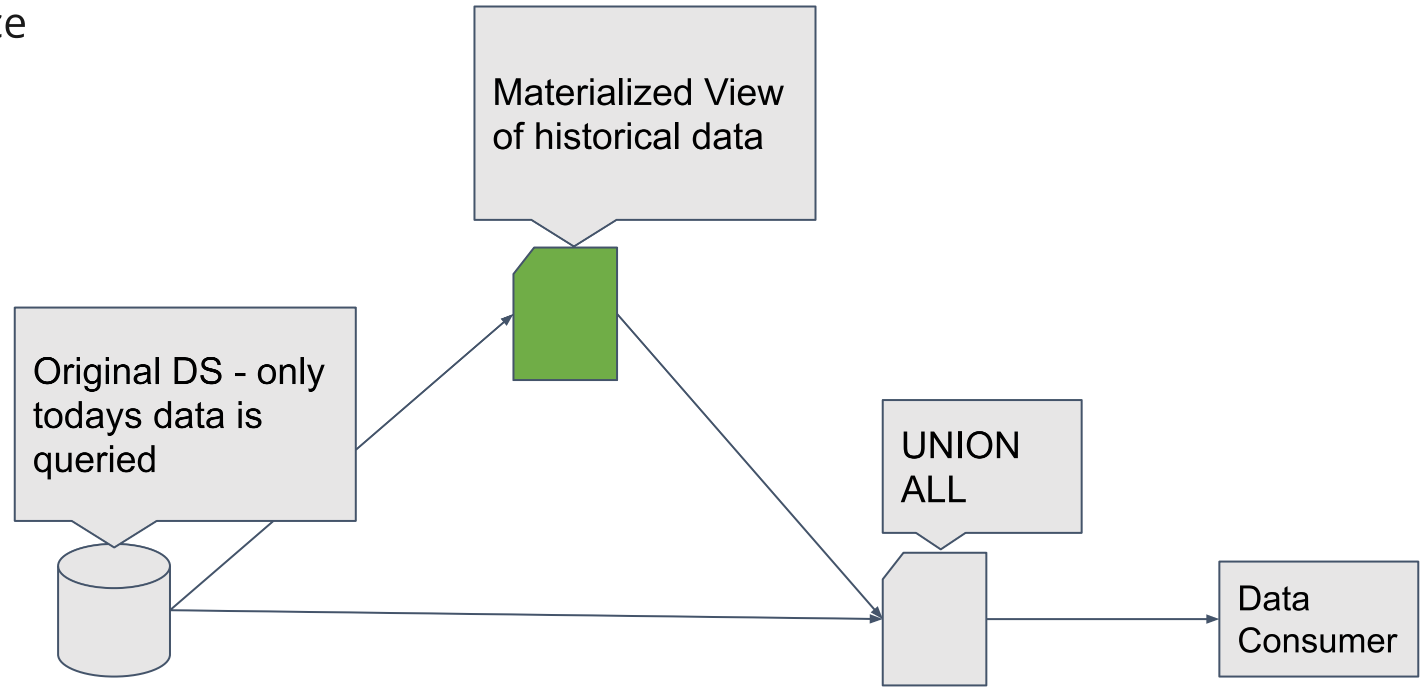


Highest performance

Lowest flexibility

Materialization Considerations - Lambda architecture

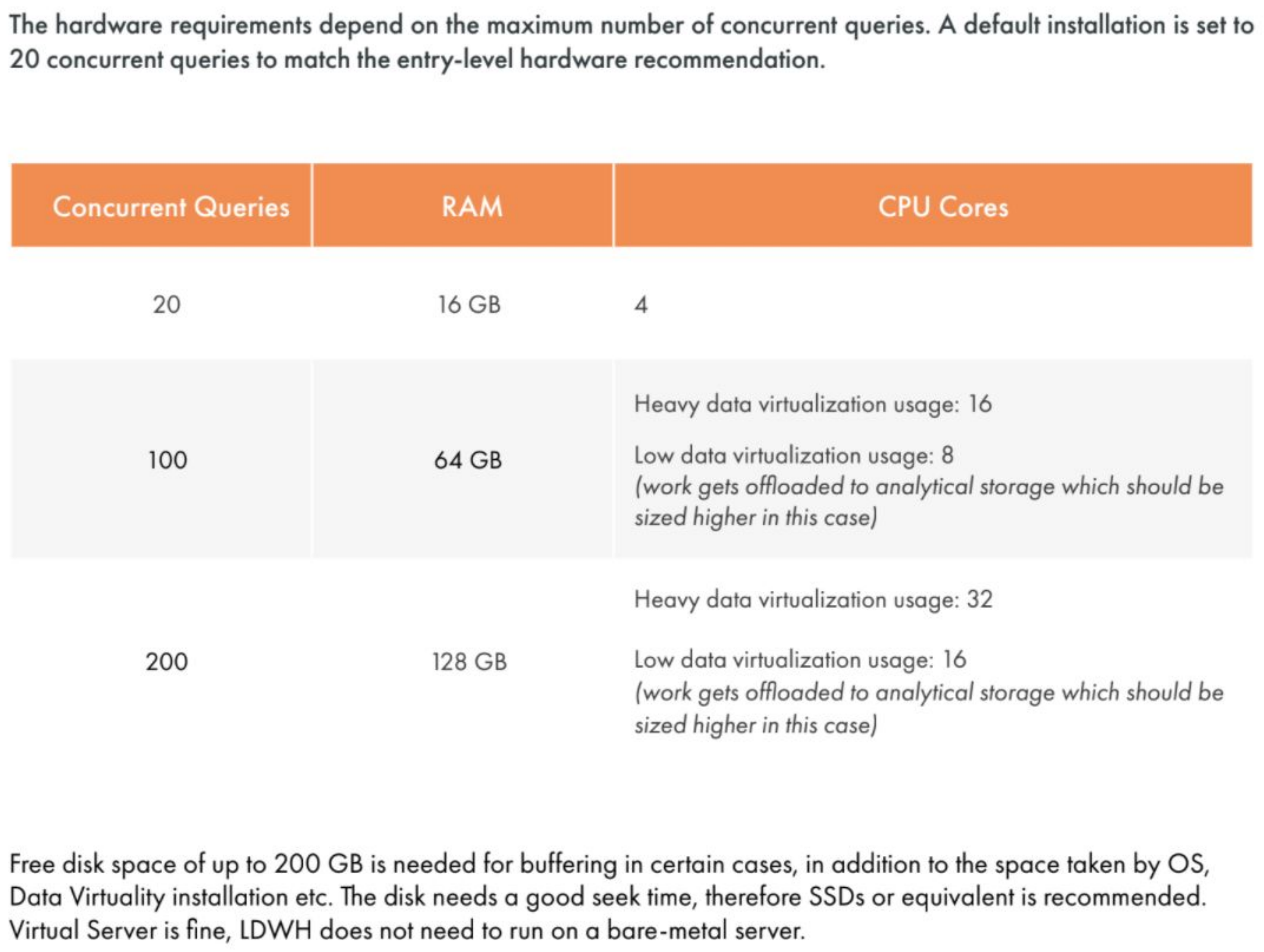
- Materialize historical data
- Union with live data
- This way we have high performance for analytical queries with historical data and low stress on the data source


Materialized View of historical data
Original DS - only todays data is queried
UNION ALL
Data Consumer

Demo - Lambda architecture

LDW Sizing

LDW Sizing



The hardware requirements depend on the maximum number of concurrent queries. A default installation is set to 20 concurrent queries to match the entry-level hardware recommendation.

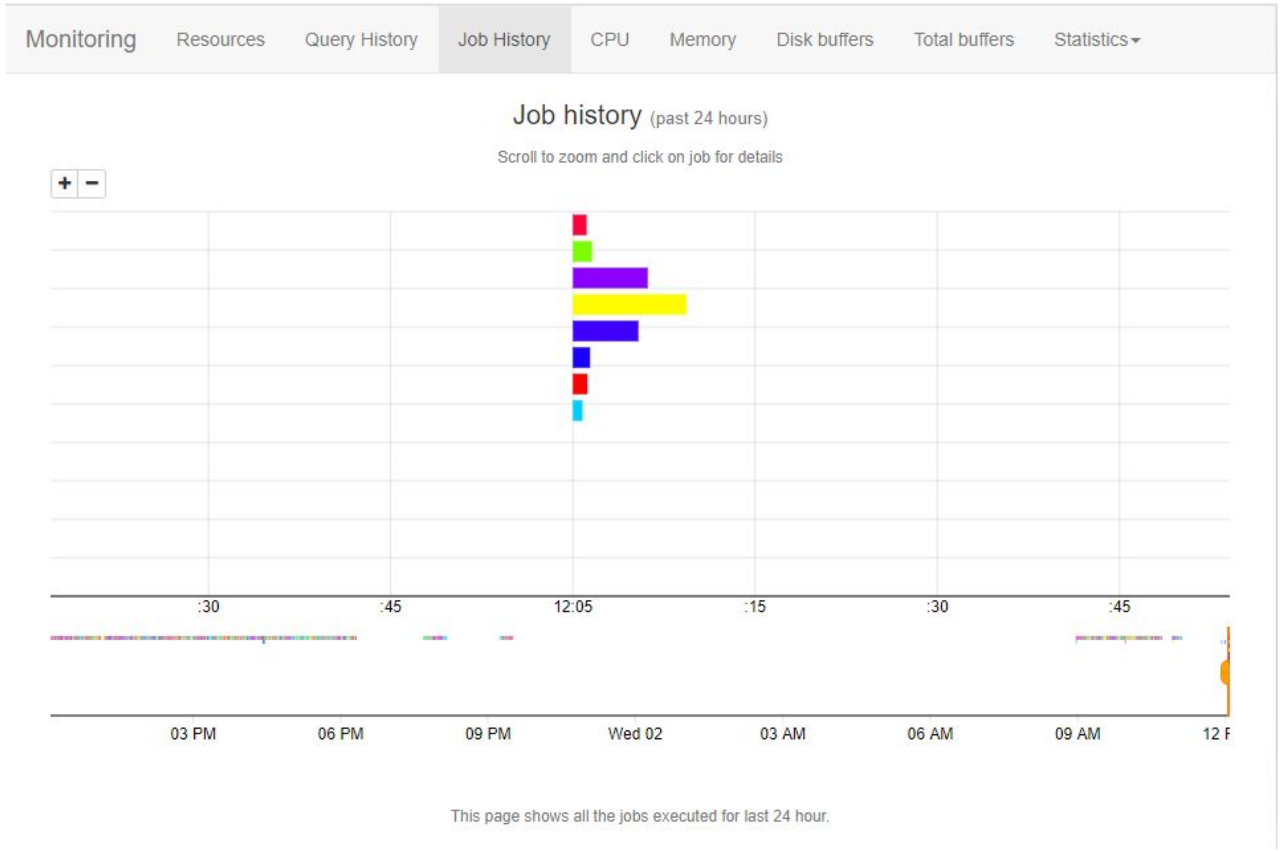
Concurrent Queries	RAM	CPU Cores
20	16 GB	4
100	64 GB	Heavy data virtualization usage: 16 Low data virtualization usage: 8 *(work gets offloaded to analytical storage which should be sized higher in this case)*
200	128 GB	Heavy data virtualization usage: 32 Low data virtualization usage: 16 *(work gets offloaded to analytical storage which should be sized higher in this case)*

Free disk space of up to 200 GB is needed for buffering in certain cases, in addition to the space taken by OS, Data Virtuality installation etc. The disk needs a good seek time, therefore SSDs or equivalent is recommended. Virtual Server is fine, LDWH does not need to run on a bare-metal server.

Identifying job overlaps

Identifying Job Overlaps

- The Monitoring tool can help assist troubleshooting issues
- here, we see all jobs started at the same time
- Job dependencies are a great way to decrease the load on both data sources and DV



This page shows all the jobs executed for last 24 hour.

Demo - Monitoring Tool

Summary

- Implementing DV's best practices helps coping with the following challenges
    - finding your metadata
    - robustness against changes
    - easier permission application
- We are aware that clients in this masterclass already implemented a lot, but with the best practices in mind, it might still be helpful for future implementations

Any feedback / questions?

Thank you!

Please feel free to contact us at:
presales@datavirtuality.com

or

visit us at:
datavirtuality.com